VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

NATIVNÍ APLIKACE PRO MOBILNÍ ZAŘÍZENÍ

NATIVE APPLICATION FOR MOBILE DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. ZUZANA BENÍČKOVÁ**
AUTHOR

VEDOUCÍ PRÁCE — **doc. RNDr. JITKA KRESLÍKOVÁ, CSc.**
SUPERVISOR

BRNO 2018

Abstrakt

Skúmaná diplomová práca sa zaoberá univerzálnou mobilnou klient-server aplikáciou Android a iOS zariadení riešenú v natívnom prostredí daných operačných systémov. Práca skúma prístupy riešenia problému v oboch prostrediach. Dané aplikačné riešenie bude zobrazovať, ukladať a spracovávať rôznorodé dáta zaslané serverom v unifikovanej podobe ako aj prispôsobovať vzhľad daným zaslaným podmienkam a zariadeniu. Cieľom práce je ukázať a porovnať natívny potenciál mobilného zariadenia ako aj navrhnúť unifikovanú aplikáciu často vyžadovanej problematiky - zobrazenie a prístup veľkého množstva uložených dát používateľovi.

Abstract

The diploma thesis deals with the universal mobile client-server application of Android and iOS devices solved in the native environment of the given operating systems. The work explores approaches to solving the problem in both environments. The application solution will display, store and process a variety of data sent to the server in a unified form as well as customize the look given the conditions addressed conditions and equipment. The aim of the thesis is to show and compare the native potential of the mobile device as well as to design a unified application of frequently asked issues - to view and access a large amount of stored data to the user.

Kľúčové slová

mobilní aplikace,nativní aplikace, swift, ios, java, android, klient-server model, UI

Keywords

mobile application, native application, swift, android, ios, java, client-server model, UI

Citácia

BENÍČKOVÁ, Zuzana. *Nativní aplikace pro mobilní zařízení*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Jitka Kreslíková, CSc.

Nativní aplikace pro mobilní zařízení

Prehlásenie

Čestne prehlasujem, že som túto diplomovú prácu vypracovala samostatne pod vedením pani doc. RNDr. Jitky Kreslíkové, CSc. Uviedla som všetky literárne pramene a publikácie, z ktorých som čerpala.

.....
Zuzana Beníčková
23. mája 2018

Poďakovanie

Ďakujem vedúcej diplomovej práce, pani doc. RNDr. Jitke Kreslíkové, CSc. za odborné vedenie, cenné rady a pomoc pri vypracovaní diplomovej práce. Ďakujem tiež firme BEN-SOFT, s.r.o. za poskytnutý priestor a pomoc pri riešení technickej časti projektu.

Obsah

Kapitola 1

Úvod

V dnešnej dobe nájdeme iba veľmi ťažko človeka, ktorý by nevlastnil mobil, prípadne tablet či iné prenosné zariadenie. Množstvo firiem preto hľadá spôsob ako vlastné systémy tejto dobe prispôsobiť a rozšíriť aj na mobilnú platformu. Hlavnými dodávateľmi hardvéru majorite populácie spolu s operačnými systémami sú firma Apple so zariadeniami ako iPhone a iPod vyrábanými výhradne pod vlastnou záštitou s operačným systémom iOS a naopak čisto voľne šíriteľný operačný systém Android vyrábaný firmou Google, ktorý sa postupom času našiel miesto na všetkým zvyšných zariadeniach. V minulosti existovali aj iné konkurujúce firmy ako BlackBerry či WindowsPhone, no v dnešnej dobe je nezmyselné o nich diskutovať [15]. Bohužiaľ, s novými technológiami a neustále sa meniacimi systémami je potrebné udržať krok a mnohé firmy preto hľadajú spôsob ako svoje zabehnuté systémy previesť do mobilných zariadení.

Projekt sa zaoberá natívnou klient-server aplikáciou, ktorá rieši častú požiadavku klientov - zobrazenie veľkého množstva dát jednoduchým spôsobom bez nutnosti naprogramovania mohutnej mobilnej aplikácie, ale využitie univerzálneho riešenia a prispôsobiť si ho vlastným potrebám. Zobrazuje pozitíva a univerzálnosť prístupu natívneho programovania na mobilnej strane. Zobrazuje tiež v aplikácii ako umožniť serveru určovanie hlavného chodu aplikácie na základe ľubovoľného systému firmy. Firemní vývojári znalí a zvyknutí na prostredie firmy môžu bez učenia sa nového jazyka, využiť potenciál aplikácie a efektívne celý systém nasadiť do mobilnej natívnej aplikácie. Hlavná logika sa deje na serverovej strane a tým pádom dovoľuje s aplikáciou pracovať dynamicky, bez potreby sústavného zasahovania do kódu aplikácie či zdĺhavého nahrávania na aplikačné úložisko, kde si ju používateľ musí aktualizovať. Bude sa teda jednať o natívnu aplikáciu pre iOS a Android zariadenia umožňujúcu klientovi prehliadať si on-line informácie, zanárať sa v nich s jednoduchým prehľadom histórie, ukladať si ich či inak s nimi pracovať. Dáta budú on-line no budú dočasne ukladané a v správnych intervaloch obnovené. V úvode práce sa zaoberáme rôznymi možnosťami vývoja aplikácii na mobilných zariadeniach, v nasledujúcich kapitolách si rozoberieme vybrané problematiky a ich riešenia v rámci iOS a Androidu, načrtneme návrh samotnej aplikácie a následne si rozoberieme implementáciu jednotlivých prvkov riešenia a ich testovanie. V závere zhodnotíme obe prostredia a ich implementačné prístupy, prínos projektu a jeho možné rozšírenia.

Kapitola 2

Vývoj mobilních aplikací

V súčasnosti je možnosť vyvíjať aplikácie na mobilné zariadenia dvomi spôsobmi. Prvý spôsob je využitie natívnych programovacích jazykov a prostredí pre vývoj, ktoré sú poskytované a vyvíjané samotnými dodávateľmi zariadení. Tento spôsob má výhody, že vďaka natívnemu prostrediu majú aplikácie prirodzený dizajn vhodný pre dané zariadenie či možnosť pristupovať k funkčným prvkom mobilu priamo z vlastného prostredia. Nevýhodou takéhoto vývoju je nutnosť programovať aplikáciu pre každé prostredie zvlášť, potreba poznať obe prostredia na profesionálnej úrovni a rovnako každú zmenu v systéme urobiť dva krát.

Naopak multiplatformné aplikácie sú vyvíjané v jednom spoločnom jazyka a následne kompilované na obe platformy. Výhodou ja zníženie nákladov na programovanie o polovicu, keďže nie je potreba písať aplikáciu zvlášť pre iOS a Android. Nevýhodou je nemožnosť pristupovať k prvkom mobilného zariadenia natívnou formou. Porovnanie medzi jednotlivými prístupmi ako natívne a multiplatformné vyvíjanie nie je v tejto práci predmetom záujmu, no je nevyhnutné ho spomenúť. Takéto porovnanie riešil napríklad Heitkötter s partnermi vo svojej práci [7].

Práca je zameraná na dvoch najväčších gigantov a teda programovanie aplikácii na iOS zariadenia a Androidy v ich natívnom prostredí. Už od roku 2016 99,6% 2.1 predaja inteligentných mobilných zariadení pripadá iOS a Androidu a tento trend sa dodnes nezmenil. Porovnanie operačných systémov publikoval v roku 2011 Goadrich a Rogers [5]. Práca opisuje najdôležitejšie dizajnové a programovacie techniky, spôsoby a komponenty, ktoré tieto prostredia ponúkajú a ktoré pre danú problematiku využijeme.

2.1 Multiplatformní mobilní aplikace

Jedným zo spôsobov vyvíjania aplikácie je multiplatformné programovanie pomocou webových aplikácií či ich hybridná kombinácia s natívnymi prvkami. Každé z uvedených riešení má svoje výhody aj nevýhody. Webové aplikácie sú programované pomocou HTML, CSS a JavaScript. Tento spôsob využíva lokálny webový server. Pomocou prehliadača v mobile simuluje samotnú aplikáciu. Tento striktne webový prístup nedovoľuje možnosť samostatnej, stiahnuteľnej aplikácie no je často krát sprístupnený pomocou URL, čím má výhodu dostupnosti. Výkon aplikácie samotnej je teda silno závislý na stabilnom a kvalitnom prehliadači mobilného zariadenia. V prípade takéhoto prístupu sa programuje iba jedna aplikácia vo forme webovej stránky prispôsobenej mobilnému rozhraniu. Mobil následne túto stránku reprezentuje ako aplikáciu, no na pozadí je spustená stále vo webovom rozhraní

Operating System	4Q16 Units	4Q16 Market Share (%)	4Q15 Units	4Q15 Market Share (%)
Android	352,669.9	81.7	325,394.4	80.7
iOS	77,038.9	17.9	71,525.9	17.7
Windows	1,092.2	0.3	4,395.0	1.1
BlackBerry	207.9	0.0	906.9	0.2
Other OS	530.4	0.1	887.3	0.2
Total	**431,539.3**	**100.0**	**403,109.4**	**100.0**

Obr. 2.1: Tabulka prodejů operačních systémů z roku 2016 (převzato, [15]).

ako stránka a je možné k nej tiež cez tento webový prehliadač ako ku stránke pristupovať. Nevýhodou takéhoto prístupu je, že aplikácia nemá prístup ku natívnym prvkom zariadenia ako kamera alebo GPS lokácia. Tieto nedostatky vyriešil hybridný spôsob vývoja, kedy je samotná aplikácia programovaná vo webovom rozhraní, ale je obalená natívnou funkcionalitou aplikácie na konkrétnom operačnom systéme. Podobne ako v čisto webovom riešení je táto aplikácia postavená ako webová stránka, ale má navyše prístup k API pre platformovo špecifické funkcie a možnosti [7]. Na kompiláciu sa využívajú externé hybridné systémy, kde najznámejším je PhoneGap, vyvíjaný Apache Cordova projektom [7].

2.2 Nativní mobilní aplikace

V úvode rozhodnutia pre natívnu aplikáciu si treba uvedomiť, že sa jedná o dva rôzne programovacie jazyky a programovacie prostredia. Pre iOS je to jazyk Swift v programovacom prostredí Xcode, pre Android je Java v prostredí Android studio. Pre možnosť produkcie výslednej aplikácie je nutnosť zaplatiť si vývojársky účet na iOS vývoj. Tento účet je v sume 99 eur na rok. Natívna aplikácia je vyvíjaná v jazykoch Java a Swift. Tieto prostredia sú navrhnuté a vyvíjané samotnými vývojármi operačných systémov a preto ich využitie napomáha k prirodzenému dizajnu aplikácie, efektívnosti, práce so súčasťami telefónu ako fotoaparát, GPS a dokumentovým systémom.

Kapitola 3

Vybrané kapitoly vývoje pro iOS

V súčasnosti sa dá za jedného z najväčších gigantov mobilnej sféry považovať Apple a jeho iPhony a iPady. Steve Jobs a Steve Wozniak spoločne priniesli absolútne prelomové a nové zariadenie, čím zmenili históriu a vzťah mobil - človek. Firma Apple bola založená spolu s Ronaldom Waynom v roku 1976, ich prvým produktom bol personálny počítač *Apple I.*, neskôr prvým firemným úspechom personálny počítač *Macintosh*.

Čím Apple a Steve Jobs zmenil svet bolo predstavenie prvej generácie mobilného zariadenia iPhone v apríli roku 2007. Do dnešnej doby bolo vydaných 11 generácii zariadenia. Okrem personálneho zážitku priniesli množstvo technologických noviniek ako napríklad plocha s detekciou rôznych či viacerých typov dotyku. Spolu s novým zariadením prišiel aj nový operačný systém iOS. V začiatkoch vydania bol systém uzavretý, nedalo sa teda vydávať vlastné aplikácie. Zariadenie malo základné aplikácie pôvodného iPodu ako hudba a kalendár, ku ktorým boli v prvom vydanom modeli iPhone-u pridané funkcie na telefonovanie a posielanie správ. Keďže Steve Jobs preferoval uzavretý systém a aplikácie tretích strán odkazoval na stavbu webových aplikácii zobrazovaných pomocou integrovaného webového prehliadača Safari, nebolo možné ešte rok od predstavenia iPhonu vyvíjať vlastné aplikácie. Teda až v marci roku 2008 bol vydaný na verejnosť pôvodne interný jazyk Objective-C, z ktorého neskôr vzišiel Swift. Objective-C bol integrovaný do prostredia Xcode, ktoré v súčasnosti podporuje kompiláciu jazykov ako C či Python vďaka modifikovanej verzii GNU Compiler Kolekcii. Od tej doby boli programátori, ktorí chceli naplniť požiadavky trhu nútení naučiť sa nové konvencie jazyka Objective-C, časom jazyka Swift.

Jazyk samotný využíva to najlepšie z rôznych programovacích odvetví ako to, že sa jedná o objektový jazyk, ktorá neumožňuje viacnásobnú hierarchiu, vďaka čomu je množstvo atypických situácii jednoducho neumožnených. Taktiež ponúka ducha funkcionálneho programovania vďaka handler či closure funkcionalite, ktoré rozoberiem neskôr v tejto kapitole. Jazykom inak pripomína klasické C či jazyk Java. V nasledujúcej kapitole si priblížime základnú štruktúru aplikácie, programovací jazyk aplikácii iOS s dôrazom na Swift, v ktorom bude programovaná technická časť projektu, opíšeme jedinečnosť a výhody prostredia Xcode a následne opíšeme spôsob práce s webovými službami, či lokálnou databázou. Predpokladom pre programovanie na iOS je hardvér, na ktorom beží operačný systém macOS, mať nainštalovaný Xcode IDE a pre možnosť publikovania taktiež vývojársky účet. Aplikácie sú finálne distribuované cez Appstore - aplikačný server pre používateľov s kontom Apple. Programovanie pre iOS vychádza z programátorského prostredia Xcode, v ktorom sa programuje najčastejšie v jazyku Swift a v Objective-C. Súčasťou programovania pre Apple zariadenia je využitie framework **Cocoa** a **Cocoa Touch**. Cocoa a Cocoa Touch je abs-

traktná vrstva operačného systému iOS, vďaka ktorej je umožnené programátorom využívať hardvérove možnosti zariadenia.

3.1 Struktura aplikace iOS

Aplikácia vyvíjaná v iOS je prirodzene vedená k vývoju s modelom MVC a teda Model-View-Controller. Prepojenie medzi vrstvami ako riadiaca logika a používateľské rozhranie sú realizované pomocou Connections. Tieto pojmy a ich realizáciu v iOS budú rozoberané v nasledujúcej kapitole.

3.1.1 MVC

Modelová časť je akékoľvek dátové úložisko cez pole, list, vlastná trieda či napríklad perzistentná databáza s využitím natívnych Core Data. Modelová časť netuší nič o používateľskom rozhraní, User Interface(UI). Objekt takejto štruktúry často krát predstavuje objekt v skutočnej realite a ukladané dáta sú obrazom skutočnosti a vytvárajú rôzne inštancie aplikácie. Vrstva používateľského UI rozhrania - View - je udržiavaná primárne v súbore s názvom *Main.storyboard*. Kontrolér je objekt typu ViewController, ktorý riadi daný View.

3.1.2 Connections

Connection umožňuje objektu prepojenie s druhým objektom na základe jeho miesta v pamäti. Connection vytvárame v prostredí Interface Builder. Inteface Builder je rozhranie v Xcode umožnujúce jednoduchú prácu s dizajnom aplikácie. Potiahnutím UI komponenty v Interface Builder s akčným tlačidlom(`ctrl`) na premennú v kóde vytvoríme Connection so zvoleným objektom. Connection môžeme vytvoriť ako dva rôzne typy.

Prvým je *Outlet*, čo môžeme považovať za ukazovateľ, referenciu na danú inštanciu objektu vo view vrstve. Vďaka outletu môžeme pristupovať, prípadne konfigurovať aktuálne nastavenia objektu vrstvy používateľského rozhrania. Druhým typom connection je *Action*, alebo akcia, čo je metóda spustená na základe udalosti vo svete ako stlačenie tlačidla, či posunutie jazdca.

Druhý typ connection využívame pre dynamické funkcie v aplikácii, niečo čo sa spustí pretože to používateľ chcel alebo nastala určitá udalosť.

3.1.3 ViewController

ViewController je základnou jednotkou podobnou ako Activity 4.1. Apple v literatúre jednotlivé ViewController objekty nazýva *scény* [9]. Scéna vyjadruje povahu triedy ViewController, pretože obsahuje všetko čo sa na aktuálnej scéne deje a nachádza. Vlastný ViewController je zdedený z triedy **UIViewController** a má na starosti nasledovné:

- Obnovovať všetky aktuálne dáta blokov UI - Views - danej scény.
- Reagovať na používateľské vstupy a interakcie.
- Nastaviť veľkosť a rozloženie UI blokov scény.
- Koordinácia navzájom s ostatnými objektami a inými scénami aplikácie.

Zobrazení View Hierarchie v iOS

Každá podtrieda triedy **UIViewController** má referenciu na dôležitú vlastnosť **UIView**. Táto referencia ukazuje na koreňovú scénu zobrazenia v hierarchii všetkých scén. V čase, kedy sa pridá tento view pod objekt *UIWindow*, je vložená tiež celá hierarchia zobrazovanej scény.

Lazy Loading Lazy loading je pojem voľne preložiteľný ako lenivé načítavanie. V našom prípade to znamená, že daná inštancia view-u nie je načítaná pokiaľ to nie je potrebné a teda pokiaľ sa prvý krát reálne nezobrazí na zariadení. Táto optimalizácia zlepšuje využitie pamäti ako aj výkon aplikácie samotnej. Spôsoby ako vytvoriť zobrazenie view v hierarchii sú dva:

- *Pomocou **Interface Builder*** - v rámci Xcode prehliadača vytvoríme inštanciu a systémovo nastavíme atribúty.
- *Programovo* - manuálne v kóde, preťažením funkcie loadView.

Životný cyklus zobrazení View Nasledujúce funkcie sú funkcie, vďaka, ktorým je možné interagovať s inštanciou **UIViewController** a jej životným cyklom.

1. *init(coder:)* - akcia zavolaná v prípade prvej vytvorenej inštancie zo súboru *Storyboard*. Storyboard je spoločný súbor všetkých zobrazovaných View objektov. Táto metóda je volaná iba raz.
2. *init(nibName:bundle:)* - inicializér pre inštanciu triedy UIViewController. V prípade, že je inštancia UIViewController vytváraná v kóde, je táto funkcia volaná raz. V opačnom prípade môže byť view zo súboru Storyboard načítavaný ako viac rôznych inštancií, v takom prípade je volaný viac krát - príkladom je navigačné okno použité v aplikácii 6.3.9.
3. *loadView* - funkcia preťažená a volaná na vytvorenie UIViewController objektu programovo.
4. *viewDidLoad* - funkcia, ktorá býva preťažovaná - opäť implementovaná pre účely aplikácie, kvôli základnej inicializácii inštancie. Táto funkcia je volaná po kompletnom načítaní View pre daný ViewController. Táto funkcia je poväčšine volaná len raz pri prvom načítaní.
5. *viewWillAppear* - funkcia volaná pri každom, aj opätovnom zobrazení daného View na obrazovke. Táto funkcia je volaná pred tým ako proces zobrazenia započne. Naopak *viewDidAppear* je zavolaný po ukončení zobrazovania danej inštancie View. Rovnako *viewWillDisappear* a *viewDidDisappear* funguje pri ukončovaní zobrazenia daného View.

3.2 Programovací jazyk pro iOS

Prvým jazykom, ktorý Apple poskytol pre nadšencov vlastných aplikácii na iOS zariadeniach bol Objective-C. Z tohoto jazyka postupne vznikol jazyk Swift, ktorý je v súčasnosti primárnym jazykom programátorov na iOS pre svoju jednoduchšiu syntax.

3.2.1 Objective-C

Objective-C je objektovo-orientovaný jazyk, vychádzajúci z jazyka C, ku ktorému bolo pridané zasielanie správ prebratých z jazyka Smalltalk. Objective-C bol vytvorený v skorých 80. rokoch 20. storočia pánmi Brad Cox a Tom Love. Bol hlavným programovacím jazykom používaným Applom na vytvorenie OS X aj iOS operačného systému.

3.2.2 Swift

Vývoj jazyka Swift započal v roku 2010. Autorom a prvým nadšencom sa stal Chris Lattner, no v jeho šľapajach rýchlo nasledovali ďalší. Mnoho vlastností jazyka bolo prevzatých z Objective-C, ako základné knižnice, funkcionality jazyka, Rust, Haskell, Ruby, Python, C#, CLU a mnoho ďalších [11]. 2. júna 2014 bol jazyk zverejnený na medzinárodnej každoročnej Apple konferencii vývojárov WWDC, spolu s dokumentáciou dostupnou na stránkach Applu a v iBooks Store [16]. Mladší vek jazyka Swift, v porovnaním s inými jazykmi, môžeme cítiť na ľahkosti a silnej univerzálnosti kódu. Jazyk je veľmi jednoduchý a využíva to najlepšie z každého z jeho predchodcov. V tejto kapitole si rozoberieme najvýznamnejšie a najzaujímavejšie prvky jazyka. Projekt sa tiež zameriava na technológie, ktoré sú očakávané pre jadro projektu.

Typy v jazyku Swift

Typy jazyka Swift môžeme rozdeliť do troch základných skupín a to: *štruktúry, triedy a enumerátory*. Každý jeden z uvedených typov môže obsahovať:

- *Vlastnosti* - atribúty typu.
- *Inicializéry* - kód, ktorý inicializuje inštanciu premennej.
- *Metódy inštancie* - funkcie viazané na daný typ, funkcie, ktoré je možné volať iba nad inštanciou daného typu.
- *Metódy statické alebo triedy* - funkcie viazané na daný typ, funkcie, ktoré je možné volať nad danou triedou.

Štruktúry označované *struct* a enumerátory označované *enum* sú vo Swifte ďaleko silnejšími nástrojmi ako vo väčšine iných jazykov. Okrem iného vedia splňovať dané protokoly a tiež môžu byť v kóde ďalej rozširované o vlastné ďalšie funkcionality. Swift samotný aj tie najprimitívnejšie typy ako *Boolean* alebo *Integer* sú v jadre deklarované ako štruktúry. Nasledujúci zoznam typov premenných je deklarovaný vo Swifte ako štruktúra. Spomínaný Boolean, potom z čísel sú to: *Int, Float, Double*. Z textu: *Text a Character* a taktiež dátové kolekcie ako *Array, Dictionary alebo Set* sú všetko štruktúry. Z tohoto nám vyplýva, že každý jeden zo štandardných typov môže byť rozšírený, spĺňať protokol a má svoje vlastnosti, funkcie a parametre.

Optionals Jednou z kľúčových vlastností jazyku Swift sú *Optionals*[9]. Optionals ponúkajú možnosť ukladať si v premennej hodnotu alebo ju tam neukladať vôbec. Táto vlastnosť sa značí znakom ? na konci deklarácie premennej. Príkladom môžeme uviesť premennú s názvom *mozno_cislo*, ktorú by bolo možné deklarovať takto `var mozno_cislo: Int?`. Takýto zápis značí, že v danej premennej sa aj po inicializácii môže opäť uložiť nil a teda

nič. Takúto premennú je v kóde nutné ošetrovať testovaním na prázdnosť zapisovanú najčastejsie `if let cislo = mozno_cislo{ code}`. Pokiaľ je dátové naplnenie zaručené je možné premennú "rozbaliť"pomocou znaku ! a teda napríklad `let cislo: = mozno_cislo!` [1] V prípade, že je z programu zaručené, že premenná bude inicializovaná vždy, je možné deklarovať premennú ako *explicitne rozbalenú* zápisom `var mozno_cislo: Int!` čím je kompilátoru povedané, aby nepožadoval explicitné rozbaľovanie ani premennú nekontroloval. Tento zápis nám nezabraňuje uložiť nil do premennej alebo ju nemať deklarovanú vôbec. Tento zápis hovorí, že kompiler nepožaduje od kódu, aby bola premenná explicitne odbaľovaná a preto je na samotnom programátorovi túto skutočnosť ošetriť alebo jej zabrániť.

Typová inference V jazyku Swift funguje typová inferencia, čo znamená, že nie je nutné pri deklarácii premennej určovať typ dátovej štruktúry. Určuje sa iba spôsob inštancie a to variabilná, teda taká, kedy je možné jej hodnoty zmeniť, zápisom *var* a konštantná zápisom *let*. Typ dátovej premennej, môže byť samozrejme určený explicitne ako bolo ukázané v predchádzajúcom príklade o Optionals alebo je jej hodnota určená na základe dát uložených do nej pri inicializácii. Napríklad `let string = "Hello World"` znamená, že v premennej string máme uložený textový reťazec a preto je premenná implicitne deklarovaná ako String.

Guard Za zmienku pri zaujímavostiach jazyka Swift určite stojí použitie *guard*. Guard predstavuje zjednodušený zápis pre rozbalenie premennej a jej kontrolu. Kontrolu, či vo vyplnenom textovom poli máme text, si teda uľahčíme nasledovne.

```
guard let text = textField.text else {
return false
} return true
```

3.2.3 Autolayout v iOS

Dôležitou súčasťou dobrého programovania v iOS je automatický systém rozloženia objektov používateľského rozhrania na obrazovke na rôznych zariadeniach, rôznych veľkostí či typov displeja. Táto vlastnosť sa volá autolayout systém. Rozloženie obrazovky je vnímané relatívne na základe vzťahov nazývaných *Constraints* a daná obrazovka a jej rozloženie je vypočítané až v čase zobrazenia, kedy už je známa výška a šírka zariadenia. Constraint je vzťah medzi dvomi objektami určujúci ich vzájomnú polohu, vzdialenosť, či závislosti na veľkosti. Samozrejme je možnosť aplikáciu vystavať absolútnymi polohami, ale tento prístup nie je rozoberaný v tejto kapitole, keďže daný spôsob nie je preferovaný. Autolayout systém je založený na *alignment rectangle*, čomu môžeme rozumieť ako obdĺžniku zobrazenia určujúcom polohu. Atribúty, ktoré je možné definovať v rámci autolayoutu sú nasledovné:

- *Height/Width* - výška a šírka obdĺžniku zarovnania.
- *Top/Bottom/Left/Right* - vzdialenosť medzi okrajom zobrazovaného trojuholníka a iným objektom.
- *FirstBaseline/LastBaseline* - hodnota väčšinou zhodná s hodnotou Top alebo Bottom, no v niektorých prípadoch, si objekt udržuje túto hodnotu inú, napríklad v prípade

[1] V takomto prípade nám nastane chyba v prípade, že v premennej *možno_číslo* nemáme dáta, ale nil).

UITextField si udržuje inú LastBaseline kvôli písmenám s nižším vykreslením ako p alebo g.

- *Trailing/Leading* - atribút určujúci pravú a ľavú vzdialenosť od iného objektu, ktorý je viazaný na typ jazyka a jeho spôsob a smer čítania. V prípade bežného európskeho čítania zľava doprava je leading vľavo a trailing vpravo, pre čítanie sprava doľava je to opačne.

Autolayout obdĺžnik je vo väčšine prípadov zhodný so samotným rámom - *Frame* - objektu. Frame je skutočne vykresľovaný objekt. Len v malých prípadoch sú tieto objekty rôzne a to, keď konkrétnu časť vykreslenia nechceme zaradiť do výpočtu autolayoutu. Nie všetky vzťahy rozloženia je nutné explicitne definovať. Na základe nastavených constraints systém zvyšné údaje automaticky prepočítava. V prípade, že mu nejaký údaj chýba alebo sú nastavené hodnoty v konflikte, je programátor upozornený chybou. Xcode samotný poskytuje množstvo jednoduchých nástrojov na pridávanie vzťahov.

3.2.4 Delegace v iOS

Delegácie je objektovo-orientovaný spôsob akým sú spracovávané *Callback* udalostí [9]. Callback je funkcia ktorá je viazaná na vykonanie určitej akcie a spustí sa vždy, keď daná akcia nastane. Samozrejme komplexnejšie objekty sú povinné reagovať príslušnou callback funkciou na viaceré udalosti. Napríklad už spomínaný UITextfield objekt reaguje ako na vloženie textu, tak aj na stlačenie tlačidla Return, ako aj na mnoho iných udalostí. Keďže nie je spôsob ako by sa mohli viaceré callback funkcie koordinovať či zdieľať dáta, bolo nutné zaviesť spoločný objekt, ktorý dané akcie obslúži. Tomuto objektu sa hovorí *Delegate*. Delegate je objekt obsahujúci obsluhu všetkých potrebných udalostí a vie teda pristupovať, manipulovať spracovávať dáta z daných callback funkcií.

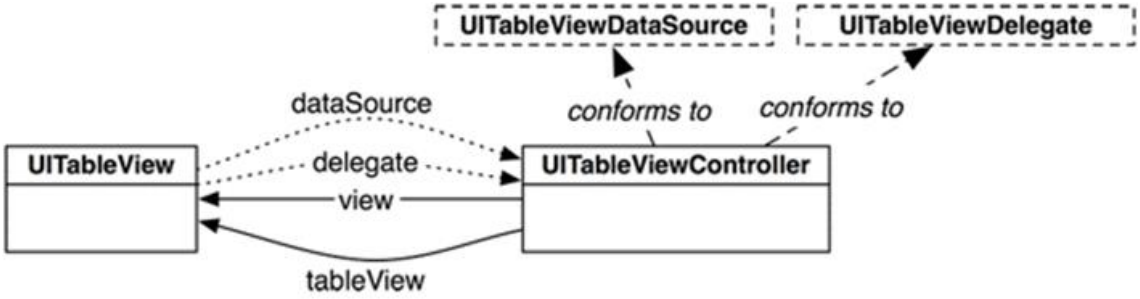
Protokol v iOS

Protokol by sa dal prirovnať k rozhraniu z jazyka Java. Je to súbor funkcií, ktoré je potrebné deklarovať v prípade, že chceme z daného objektu vytvoriť následníka, prípadne delegáta. Napríklad bez definície funkcií UITextField protokolu by nebolo možné spracovať udalosti s daným textovým poľom ako napríklad filtrovanie zoznamu pri vložení textu do textového poľa používateľom. Aby bolo možné pristúpiť k vloženým dátam - napísaný text, ktorý chceme filtrovať, je potrebné definovať funkciu callback pri akcií vloženia textu a taktiež funkcie vykonané pri všetkých ostatných možných udalostiach.

3.2.5 Closures

Closure je prvok veľmi podobný klasickým metódam a funkciám. V skutočnosti aj metódy aj funkcie sú špeciálnymi prípadmi Closure [9]. Clusures majú veľmi jednoduchú syntax vhodnú na vloženie ako vstup, či dokonca výstup funkcie.

```
{ (arguments) -> return type in
code
}
```



Obr. 3.1: Vztah mezi UITableView a UITableViewController (převzato, [9])

3.2.6 Tabulkový přehled v iOS

Jedným z prvkov používateľského rozhrania je **UITableView**. Tento prvok je jedným z kľúčových prvkov väčšiny aplikácii pretože tabuľka jednoducho zobrazuje poskytované dáta v klasickej alebo vlastnej zobrazovanej bunke tabuľky. Pre správny chod tabuľky je potrebné nastaviť jej primárne funkcie ako počet jednotlivých sekcií a riadkov, ich vzhľad, prípadne extra funkcie pri editácii. Aby bolo možné túto tabuľku použiť, je potrebné definovať podobne ako v modeli MVC všetky vrstvy tabuľky. Jej *dáta* ako modelovú vrstvu zodpovedajúce sa príslušnému protokolu **UITableViewDataSource**, následne objekt, ktorý bude obsluhovať všetky UI komponenty a predstavuje ho trieda **UITableViewController** a nakoniec delegáta, ktorý informuje ostatné objekty o uskutočnených udalostiach tabuľky. Delegát môže byť ľubovoľný objekt spĺňajúci protokol **UITableViewDelegate** [9]. Celý vzťah môžeme sledovať na obrázku 3.1. Vďaka funkcii recyklujúcej objekty buniek tabuľky `dequeueReusableCellWithIdentifier` je zabezpečená efektívna správa objektov a pamäte. V kapitole o Androide 4.2.6 uvidíme, že je zavedený podobný princíp, ale je nutné vytvárať zvlášť jednotlivé triedy na zobrazenie používateľského rozhrania a prepojenie s obsluhou naplnenia tabuľky.

3.2.7 Práce s webovými službami v iOS

Každý jeden webový prehliadač využíva HTTP na komunikáciu s webovým serverom. Na tento účel sa využíva *request* na server, ktorý špecifikuje URL. URL je špecifická adresa požadovanej stránky, ktorú na serveri chceme osloviť. Pomocou URL je možné nielen stránku osloviť, ale vďaka parametrom *query items* aj nastaviť požadované dáta, ako napríklad ich formát. Server následne odpovedá zaslaním požadovanej stránky (typicky vo forme HTML a obrázkov), ktorý vie prehliadač naformátovať a zobraziť [9]. Vo vypracovaných aplikáciách je očakávaná odpoveď vo formáte JSON. Vďaka dlhoročnej popularite webových prehliadačov je technológia HTML dobre a stabilne vyvinutá. Vďaka tomu je HTTP komunikácia prijateľná pre väčšinu firewall systémov, webové servery sú kvalitne zabezpečené s vynikajúcimi výsledkami a samotná práca s vývojom sa stala veľmi jednoduchá pre vývojára. Aby bolo možné využiť webové služby ponúkané serverom je potrebné implementovať HTTP komunikáciu.

Na zbalenie všetkých informácii o nastávajúcej komunikácii iOS používa triedu **URLRequest**. Najdôležitejšou súčasťou tohoto objektu je špecifikovaná URL. Ďalšou dôležitou súčasťou webovej komunikácie je **URLSession** API. URLSession API je súbor tried, ktoré sú potrebné, aby *request* vedel komunikovať so serverom viacerými spôsobmi. Trieda **URL-**

SessionTask je zodpovedná za komunikáciu so serverom a samotná trieda **URLSession** je zodpovedná za vytvorenie korektného URLSessionTask na základe danej konfigurácie [9].

JSON a JSON v iOS

JSON, alebo JavaScript Object Notation, je formát zasielaných dát, ktoré môže obsahovať základné typy ako:

- pole
- slovník
- text
- číslo

Jeho štruktúra je postavená na jednej alebo viacerých dvojiciach kľúč-hodnota. Kde *hodnota* môže byť opäť ľubovoľný typ. Dokument začína aj končí kučeravými zátvorkami - {}, ktoré značia slovník, hranaté zátvorky pole - [] a úvodzovky značia "text". Na spracovanie JSON súborov sa využíva trieda **JSONSerialization**, ktorá z JSON súboru vytvorí adekvátne objekty na základe typu.

3.2.8 Lokální databáze v iOS

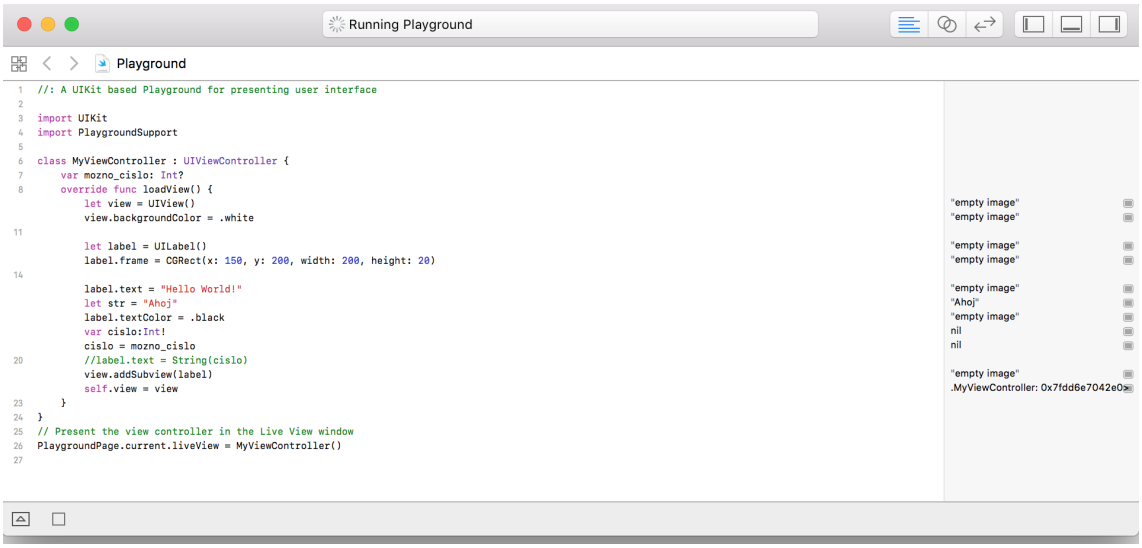
Pokiaľ je požiadavka na aplikáciu uchovávať dáta lokálne a trvalo, máme v rámci iOS dve možnosti. Jednou z nich je využitie Archivovania súboru s dátami do súborového systému. Táto možnosť je vhodná na riešenie, kedy nám dáta stačí raz načítať a následne raz uložiť. Bohužiaľ, ale nie je možnosť pracovať s týmto súborom čiastkovo a teda načítať iba časť dát alebo zmeniť iba jeden záznam. Tieto nedostatky rieši povaha lokálnej databázy Core Data. Core Data ponúka prácu s dátami ako získanie, mazanie, zmenu či vkladanie dát parciálne do súboru. Využitie Core Data môže radikálne ovplyvniť výkon aplikácie pokiaľ potrebujeme zdieľať veľké množstvo modelových objektov medzi RAM pamäťou a súborovým systémom [9]. Core Data nie je jedinou možnosťou, ktorá sa dá na uloženie dát použiť. Vždy je možnosť siahnuť po externých platformách ako napríklad databáza Realm.

Core Data

Core Data je framework – aplikačný rámec, na perzistentné uloženie dát, vydané Applom pre macOS ako aj iOS operačný systém. Core Data umožňujú uložiť dáta ako relačný entita-atribút model, ktorý je následne serializovaný ako XML súbor, binárny súbor alebo ako SQLite úložisko. Úvodné vytvorenie modelu je veľmi podobné relačnej databáze a teda sa nastavujú:

- *Entity*, a jej *atribúty*, ktoré predstavujú tabuľky v relačnej databáze.
- *Vzťahy (Relationships)* medzi *entitami*, predstavujúce nadväznosti cudzích a primárnych kľúčov.

Najväčším rozdielom od klasických relačných databáz je spôsob práce s dátami. Na prácu s dátovým modelom sa využíva trieda **NSManagedObject**, kde z každej *entity* je vytvorená podtrieda s ktorou pracujeme. Od logiky na pozadí ako sú jednotlivé entity prepojené je opäť programátor odbremenený a pracuje v objektoch s entitami ako s vlastnými atribútmi.



Obr. 3.2: Playground nabízen Xcode (snímek obrazovky)

3.3 Prostředí iOS

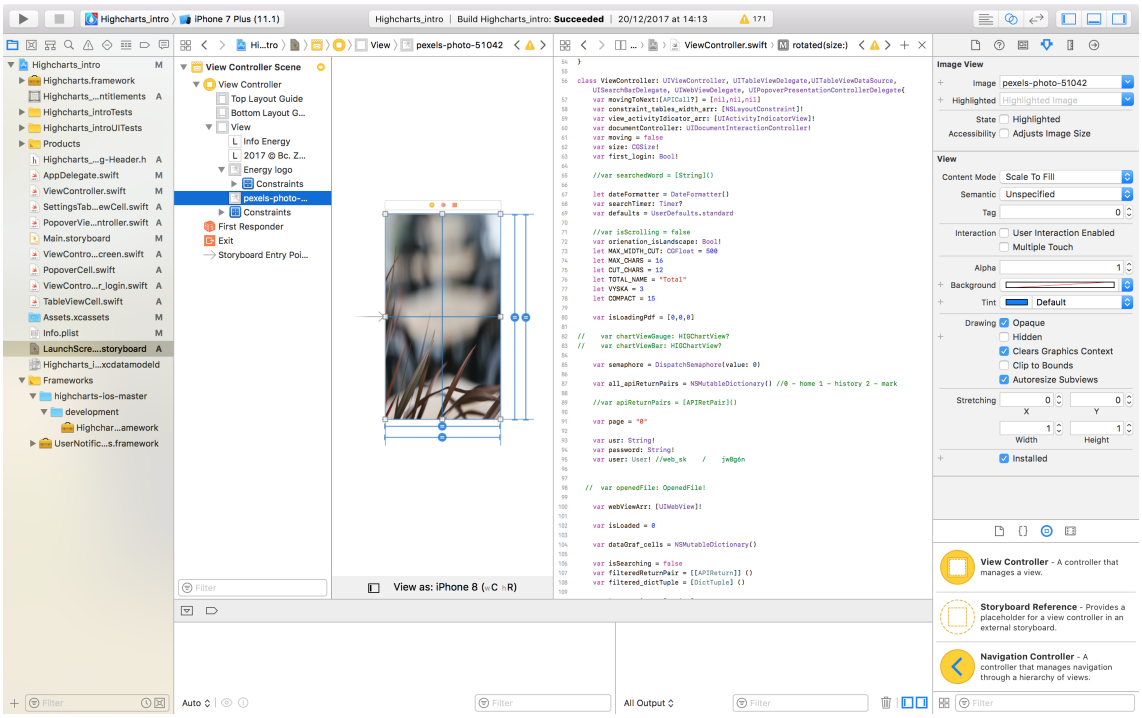
V kapitole sú rozoberané zaujímavosti z prostredia, ktoré je primárne pre programovanie iOS. Týmto prostredím je Xcode, ktorý je možné voľne stiahnuť v Appstore - aplikačnom serveri poskytovanom Apple spoločnosťou.

3.3.1 Xcode

Prostredie Xcode je vývojárske prostredie, v ktorom je možné vyvíjať aplikácie s operačným systémom macOS, iOS, watchOS a tvOS. Prvý krát mali možnosť programátori vytvoriť vlastnú Apple aplikáciu v roku 2003 a v súčasnosti je aktuálne najnovšia verzia 9.2. Podporuje jazyky ako C,C++, Objective-C, Python, Ruby, Swift a iné. Vďaka využitiu technológie Match-O formátu, ktorý obsahuje kód pre viaceré architektúry, je možné, aby boli aplikácie spustené na PowerPC aj Intel platforme. Spolu s Xcode-om bol predstavený tiež softvér *Instruments*, ktorý aplikáciu vizualizuje a analyzuje. Podobne v Android umožňuje nástroje Lint 4.3.1. Xcode od verzie 4.1 obsahuje GNU Compiler Collection. Ponúka tiež možnosť Playgroundu, ktorý v čase kompiluje a zobrazuje výsledné a aktuálne hodnoty pre jednoduché testovanie či výuku 3.2. Taktiež možnosť aktuálne zobrazeného dizajnu, či prehľadu dvoch okien 3.3.

3.4 Dizajn iOS aplikací

V dnešnej dobe najpoužívanejšie aplikácie ako Facebook zastávajú vlastný dizajn na oboch platformách a v ich prípade je teda nepodstatné hovoriť o dizajne závislom na operačnom systéme [1]. Je možné, že s postupnou narastajúcou popularitou programovania multiplatformovo bude tento trend postupne narastať a aplikácie budú v budúcnosti všetky vedené rovnakým dizajnom. Napriek tomu prirodzené používanie komponentov ako aj odporúčania Apple sú dodržiavať základné princípy, ktorými sa snažia oddeliť od ostatných a tými sú:



Obr. 3.3: Xcode pracovní plocha v Assistant editor módu, Interface Builder a kód (snímek obrazovky)

- *Jasnosť* Pomocou konzistentných farebných kombinácii, veľkostí a zreteľnosti písmen, ikon a správnych textov. Aplikácie sú na pohľad rozoznateľné ako iOS vďaka čistému, vzdušnému dizajnu.
- *Ohľad* Plynulý pohyb a ostré, jasné, prehľadné rozhranie napomáha ľuďom pochopiť a komunikovať s obsahom napriek tomu, že s ním nemuseli mať v minulosti žiadne skúsenosti [8]. Obsah by mal vždy vyplniť celú obrazovku a rozmazaním alebo priehľadnosťou sa snažíme používateľa naviesť alebo mu ukázať viac z obsahu.
- *Hĺbka* Rozdielne vizuálne vrstvy a realistický prechod medzi scénami vyjadrujú hierarchiu, dodávajú aplikácii sviežosť a významne napomáhajú pochopeniu zo strany používateľa a dávajú mu pocit hĺbky aplikácie.

3.5 Shrnutí iOS vývoje

Programovanie v iOS prostredí prináša eleganciu a ľahkosť v navrhovaní dizajnu aplikácie. Je v mnohých smeroch jednoduchšie spraviť niektoré veci, ako napríklad využitie Connections v prostredí Builder bez hlbšieho pochopenia logiky na pozadí. Naopak rovnako ako je zvykom u používateľov mobilov, v Androide má programátor voľnejšiu ruku, ale je nútený konfigurovať väčšie množstvo nastavení ručne. Najväčšou nevýhodou programovania pre Apple zariadenia je nutnosť investovať do zariadenia a tým pádom horší prístup pre programátora začiatočníka či učebnú skupinu.

Kapitola 4

Vybrané kapitoly vývoje pro Android

Android je mobilný operačný systém, vyvinutý pôvodne firmou Android Inc., ktorú v roku 2005 odkúpil Google. Android je založený na modifikovanej verzii Linuxového jadra a iných open source-ových softvérov. Na rozdiel od iOS vývoj na Android je platformovo nezávislý a nevyžaduje žiadny platený účet či iný poplatok. Voľnosť vývoja na tento operačný systém je dôvod, prečo sa teší veľkej popularite medzi vývojármi. Taktiež jazyk Java, v ktorom je Androidová aplikácia písaná, je omnoho rozšírenejší ako Objective-C alebo Swift. Android tiež ponúka vývoj na Android TV, Android Auto a Android Wear. Rôzne varianty Androidu sú využívané aj inde ako na mobilných zariadeniach ako napríklad na hracích konzolách, digitálnych kamerách, počítačoch či inej elektronike. Aplikácie sú finálne distribuované pomocou Goole Play Store, no na rozdiel od Apple zariadení je možné si v nastaveniach povoliť možnosť vlastného sťahovania APK - aplikačných súborov, voľne šírených internetom na vlastnú zodpovednosť. Používateľ riskuje vlastnú bezpečnosť zariadenia, no je zachovaná voľnosť a sloboda rozhodnutia používateľa a opäť zjednodušený vývoj a distribúcia vlastných aplikácii.

V tejto kapitole si rozoberieme najvýznamnejšie črty programovania pre Android, typický dizajn Android aplikácii a teda spôsoby, ktoré budú počas návrhu aplikácie dodržiavané.

4.1 Struktura Android aplikace

Každá z Android aplikácii sa skladá z troch hlavných komponentov, ktorými sú: Aktivity, Zdroje a Manifest. *Aktivity* predstavujú aktívne, programové časti funkcie, *zdroje* statické zdrojové súbory aplikácie ako obrázky, texty, rozloženia komponent. Rovnako ako v iOS prípade je aplikácia vedená ku modelu MVC 3.1.1 a teda Model-View-Controller. Objekty vrstvy modelu sú všetky triedy, ktoré v aplikácii predstavujú dátové úložisko. *Modelovú vrstvu* tvoria triedy, v ktorých ukladáme dáta definujúce povahu aplikácie. Modelová Časť nevie nič o UI komponentoch či logickom chode aplikácie. *View* v našom prípade Androidu **Layout** je XML súbor, ktorý definuje všetky UI komponenty aplikácie. Nakoniec funkcia *kontroléru* v úlohe **Aktivít**, ale aj ďalších pokročilých tried ako **Fragmenty** či **Servisy**. Základná, jednoduchá štruktúra aplikácie pozostáva z *activity a layout*, kde:

- *Activity* je inštancia triedy knižnice Android SDK zdedená z triedy **Activity**. Trieda je zodpovedná za spracovanie podnetov z vrstvy používateľského rozhrania UI a teda

všetka interakcia používateľa, ako stlačenie tlačidla, prípadne plynutie času (ukončenie časovača). Odpovedá na otázku, *Čo nastane, ak nastane táto udalosť?* Je veľmi podobná s triedou ViewController 3.1.3 v iOS aplikáciách.

- *Layout* môžeme definovať ako súbor určených definícií rozmiestnenia objektov UI. Layout je určený pomocou súboru XML definícii. Znalosť XML je predispozíciou pre ďalšiu prácu s Androidom, preto sa s jeho znalosťou ráta ako u čitateľa tak aj u nádejného programátora Android aplikácií. Každá definícia v XML predstavuje jeden z blokov UI prostredia ako tlačidlo prípadne text, ktoré nazývame *widget* [13].

4.1.1 Životní cyklus třídy Aktivita

Rovnako ako bol v iOS preberaný životný cyklus zobrazenia View triedy 3.1.3 v Androide je možné diskutovať o životnom cykle triedy Aktivita. Počas jedného životného cyklu sa trieda môže dostať do štyroch rôznych stavov a to:

- bežiaca
- pauzovaná
- stopnutá
- neexistujúca
- *schovaná(dead)*

Každá jedna zmena stavu aktivity má metódu, ktorá aktivitu upozorňuje na túto zmenu [13].

4.1.2 Widget

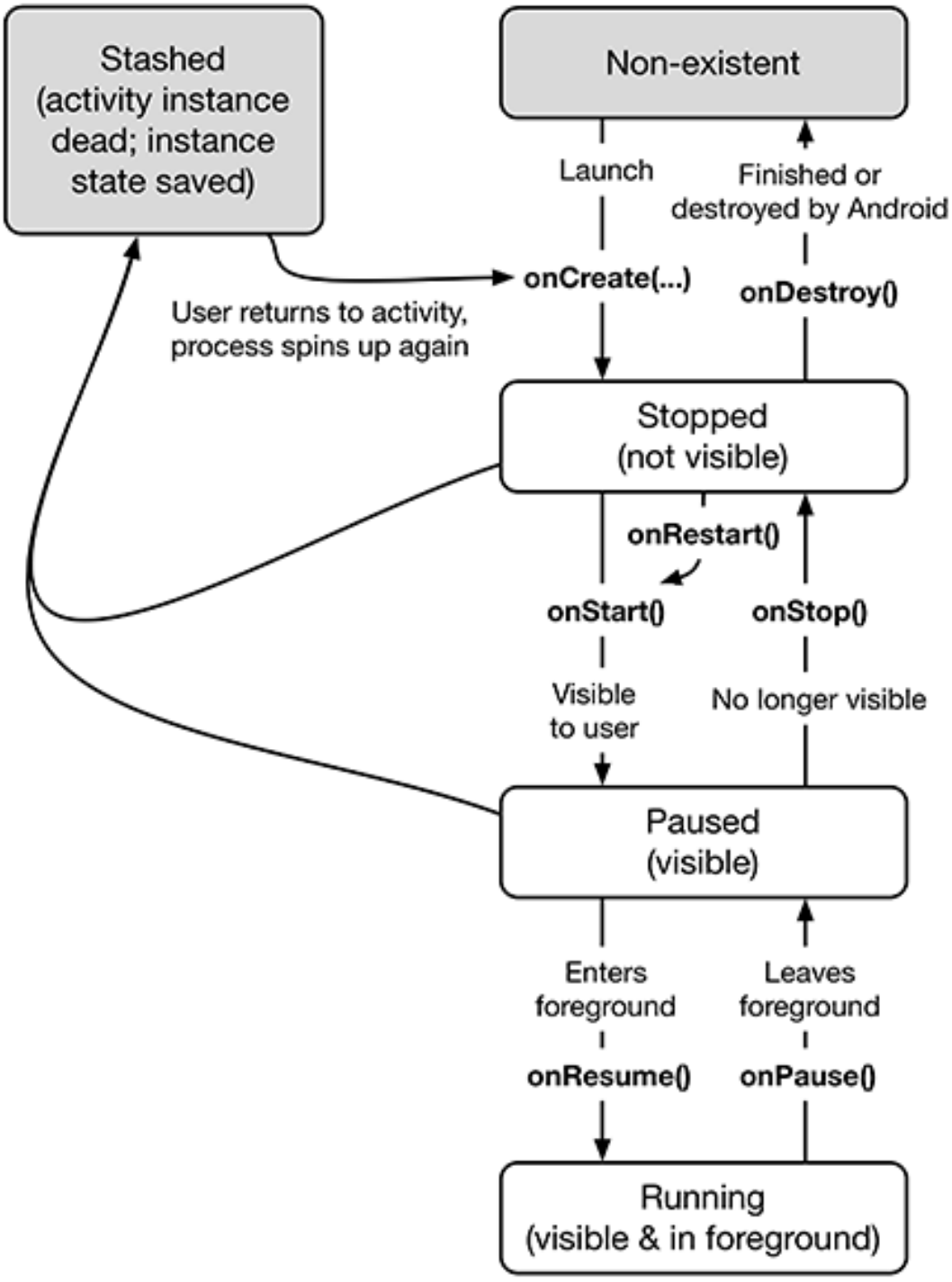
Widget je základný blok, pomocou ktorého staviame UI. Widget je univerzálny objekt, ktorý môže zobrazovať text či inú grafickú reprezentáciu, interagovať s používateľom, môže určovať pozície jednotlivých blokov, predstavovať tlačidlo, vstup používateľa prípadne zaškrtávacie políčko a iné. Každý z takýchto objektov widget je možné použiť v rámci aplikácie a prispôsobiť ho vlastným potrebám. Každá z takýchto tried bude zdedená z triedy **View** alebo jej podtried ako napríklad **TextView** alebo **Button** [13]. Pre objekt widget je potrebné nastaviť parametre ako výšku a šírku, ktoré sa najčastejšie nastavujú pomocou automatických parametrov prispôsobujúcich sa nadradenému prvku alebo potrebnej výške. Sú to:

- match_parent
- wrap_content

Samotný koreňový View nazývaný **LinearLayout** má nastavený match_parent pre View, ktorý mu je priradený ako rodičovský - nadradený, a v ktorom aplikácia existuje [13].

4.1.3 Propojení modelové a kontrolní části

V nasledujúcej kapitole rozoberiem ako prepojiť Layout a jednotlivé Widgety, vizuál s funkčnou, obsluhujúcou časťou. V iOS sa na tento účel využívajú Connections 3.1.2. Layout tak ako aj iné súbory projektu, ktoré nie sú konkrétne kusy kódu, ako obrázky, XML súbory,



Obr. 4.1: Zobrazení životního cyklu třídy Aktivita v Androidu (převzato, [13]).

audio obrázky a iné, sa nazývajú *Resources - zdroje*. Tieto súbory sú ukladané v zložke *app/res*. Aby bolo možné sa k takýmto zdrojom dostať aj v rámci kódu, je potrebné, aby sa k ním pristúpilo pomocou *Resource ID*.

Prístup k takýmto zdrojom značne uľahčuje trieda auto generovaná Androidom a to *R.java*, vďaka ktorej môžeme pristupovať ku hľadanému ID objektu pomocou volania *R.layout.NAZOVOBJEKTU*. Rovnako ako layout aj string - textový záznam je potrebné ukladať v takomto externom priečinku. Android vďaka tomu môže pre layout a stringy automaticky generovať ID a nie je potrebné vytvárať ich samostatne. Toto však neplatí pre jednotlivé *widget*, nakoľko pri nich nie je zaručené, že k nim budeme potrebovať pristupovať z kódu. Musíme teda v rámci XML definície určiť ID definovaného widgetu takýmto zápisom *android:id="@+id/NAZOVID"*. Následne je možné z kódu takýto objekt priradiť premennej predstavujúci daný widget pomocou funkcie `public View findViewById(int id)` v Jave, prípadne v Kotline je možnosť pristupovať priamo. K vytvorenému ID sa pristupuje opäť cez R triedu a to takto *R.id.NAZOVID*. Môžeme vidieť, že na rozdiel od Connections v iOS, kde je nutné prepojiť objekt s premennou graficky potiahnutím a logika za prepojením týchto objektov sa odohráva na pozadí Xcode, v Androide prepájame tieto objekty ručným vlastným pomenovaním a následným priradením premennej pomocou funkcie v kóde.

4.1.4 Manifest

Ďalšou z nevyhnutných vecí pri programovaní Android aplikácií je potreba každú z Aktivít uviesť z XML súbore *Manifest*. K tomuto súboru pristupuje operačný systém, pretože obsahuje všetky potrebné metadáta o aplikácii, ktoré na chod systému potrebuje. Okrem iných vecí Manifest má na starosti nasledovné:

- Pomenováva Java balík, ktorý slúži ako jedinečný identifikátor aplikácie.
- Opisuje všetky komponenty aplikácie ako aktivity, servisy a iné.
- Určuje procesy, ktoré sa môžu zúčastniť aplikačných komponentov.
- Definuje práva a povolenia na prácu s konkrétnymi časťami hardvéru či inými aplikáciami.
- Definuje najnižší možný operačný systém cieľového zariadenia a iné.

4.2 Programovací jazyk Android

Aplikácie pre Android sú písané pomocou Android SDK - Software Development Kit (súbor vývojárskych nástrojov ako knižnice, debugger, emulátor, tutoriál) a jazyka Java. Kedysi bol primárnym prostredím pre vývoj Eclipse, no v súčasnosti je odporúčaným nástrojom Android Studio. Novinkou od roku 2017 je tiež jazyk Kotlin. V tejto kapitole si rozoberieme programovacie techniky Android aplikácii a najpoužívanejšie komponenty, ktoré neskôr v projekte využijeme. Aplikácie sú programované v jazyku Java a využívajú XML technológiu.

4.2.1 Kotlin

Kotlin je nový jazyk vyvinutý firmou JetBrains. Je navrhnutý pre programovanie multiplatformných a Android aplikácii. Je 100% kompatibilný s Java súbormi a prináša výhody,

oproti Jave, podobné ako priniesol Swift oproti Objective-C. Novinkami sú optionals 3.2.2, typová inferencia 3.2.2, prednastavené hodnoty funkcií (menej preťažených rovnakých funkcii), priamy prístup ku prvkom bez `findById` a iné. Výhody jazyka Kotlin sa podobajú výhodám jazyku Swift a preto môžeme predpokladať, že sa jazyk stane v blízkej dobe populárnym.

4.2.2 Java

Java je objektovo-orientovaný jazyk, vyvíjaný spoločnosťou Oracle. Bol iniciovaný Jamesom Goslingom a partnermi v roku 1991. Je to jeden z najrozšírenejších jazykov pre svoju platformovú či hardvérovú nezávislosť. V súčasnosti je najnovšou verziou Java 9. Jedným z hlavných cieľov jazyka Java je prenositeľnosť. Znamená to, že jeden krát naprogramovaný kód je kompatibilný s rôznymi kombináciami hardvéru a operačného systému. V projekte bol zvolený vývoj Javou práve pre jej spoľahlivosť a univerzálnosť.

4.2.3 Práca s webovými službami v Androidu

Rovnako ako v prípade iOS 3.2.7 je základným objektom **URL**. Potrebné funkcie *request* požiadavky na server zaobstaráva objekt **URLConnection**, prípadne v HTTP komunikácii je možné ho pretypovať na konkrétnejší **HttpURLConnection**, ktorý ponúka špecifickejšie metódy na prácu s HTTP komunikáciou. Návratom takejto požiadavky bude (v našom prípade) formát JSON, ktorý je rozoberaný už v kapitole o iOS 3.2.7. Android pristupuje ku JSON formátu pomocou triedy **JSONObject**.

4.2.4 Lokálna databáza v Androide

V Androide je primárnou databázou, ktorá sa využíva, relačná databáza **SQLite**. Jednotlivé tabuľky a riadky sa vytvárajú pomocou vlastnoručne vytvorených tried. Android umožňuje pomocné triedy ako **SQLiteOpenHelper**, ktorá zaobstaráva základné potrebné procesy ako:

- Kontrola, či databáza existuje.
- Pokiaľ neexistuje, jej vytvorenie aj s inicializačnými dátami.
- Pokiaľ existuje, otvorenie a kontrola verzie modelu.
- Pokiaľ je verzia stará, update na aktuálnu verziu.

Samotný objekt databázy obstaráva trieda **SQLiteDatabase**.

SQLite

Relačná databáza zameraná na ukladanie perzistentných modelov dát do zariadenia, naopak od klient-server databázy ako SQL. SQLite je obsiahnutý v relatívne malej knižnici napísanej v C. SQLite vychádza z SQL štandardu a rovnako dodržiava všetky zásady ACID prístupu ako : atomicita, konzistencia, izolácia a odolnosť.

4.2.5 Možnosti automatického rozložení obrazovky

Aplikáciu môžeme prispôsobiť rôznym veľkostiam obrazoviek zariadení tromi rôznymi spôsobmi. V rámci triedy **ViewGroup**, ktorá je nadradenou, rodičovskou triedou zobrazovaných komponentov, je možné usporiadať UI objekty nasledovne:

- *lineárne* - Objekty sú usporiadané do horizontálneho alebo vertikálneho listu objektov a je umožnené posúvanie po tabuľke v prípade, že obsah presiahol veľkosť obrazovky zariadenia.
- *relatívne* - Podobne ako v iOS 3.2.3.Vzdialenosť je relatívna na základe ostatných objektov.
- *webovo* - Klasický webový blokový prístup zobrazujúci HTML stránky [4].

4.2.6 Tabulkový přehled v Androidu

Podobne ako v programovaní pre iOS, je základným komponentom zobrazovania dát tabuľka. V Android prípade sa tabuľka zostavuje z tried: **RecyclerView, Adaptér** a **ViewHolder**. RecyclerView umožňuje zobraziť listový zoznam objektov a je podtriedou triedy **ViewGroup**. V závislosti od zložitosti zobrazovanej bunky môže byť zobrazovaná trieda veľmi jednoduchá alebo komplexná. Z názvu rozumieme, že aj pri väčšom množstve riadkov v tabuľke systém objekty nevytvára, ale opäť recykluje už vytvorené. Podobný princíp funguje v iOS systéme. 3.2.6. Úlohou RecyclerView-u je teda zabezpečiť vytvorenie určeného počtu recyklovateľných objektov triedy **ViewHolder**. ViewHolder je trieda zodpovedná za zoskupenie a udržovanie všetkých objektov **View** triedy, ktorá sa v danej bunke bude nachádzať. Táto trieda predstavuje jeden riadok tabuľky. Na správu a naplnenie riadkov RecyclerView využíva *adaptér* [13]. Adaptér je zodpovedný za:

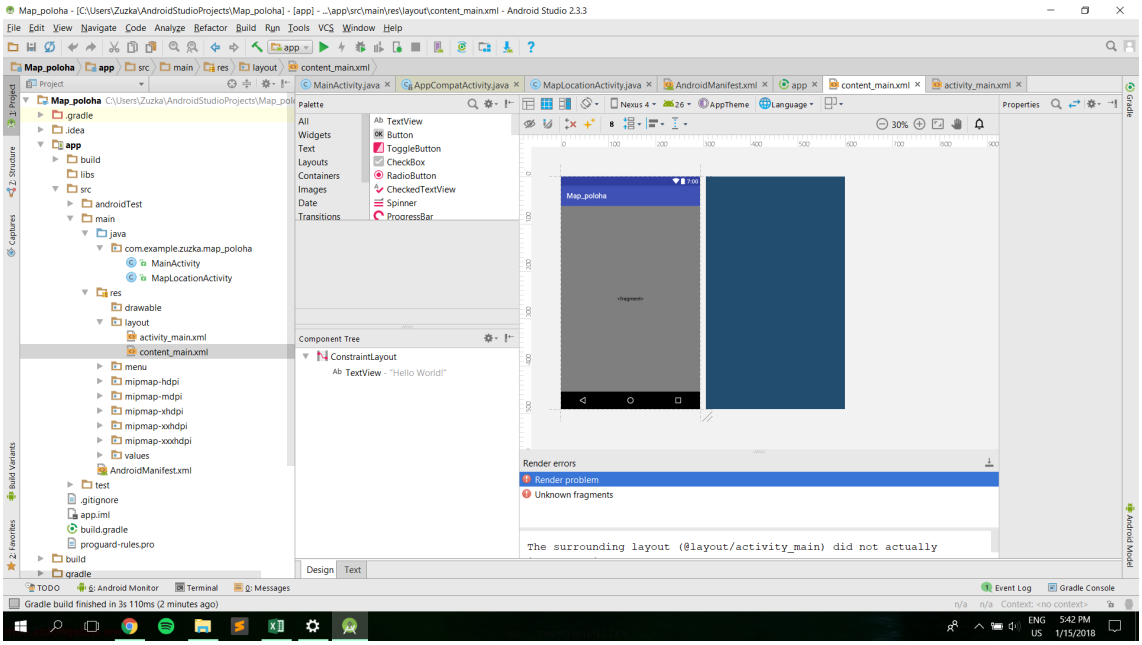
- Vytvorenie potrebných **ViewHolder**.
- Napojenie **ViewHolder** objektu na potrebné dáta modelovej vrstvy.

4.3 Prostředí Android

Výhodou programovania pre Android je voľná ruka pri výbere platformy vlastného zariadenia. Programátor môže pracovať vo svojom obľúbenom prostredí čím zvyšuje svoju efektívnosť. Vďaka tomu nie je potrebná prvotná investícia do hardvéru a rovnako nie je potrebný nákup licencie na vývoj. V minulosti bolo Googlom preferované programátorské prostredie Eclipse, ktoré je dodnes veľmi populárne a rozšírené. Napriek tomu prišiel Google v roku 2014 s odporúčaným primárnym prostredím Android Studio.

4.3.1 Android Studio

Android Studio je platformovo nezávislé, integrované vývojárske prostredie vyvinuté Googlom, ktoré je primárnym nástrojom pre vývoj aplikácii na Android zariadenia. Primárne je založené na princípe IntelliJ IDEA. IntelliJ IDEA je vývojárske prostredie s jazykom JAVA vyvinuté spoločnosťou JetBrains od čoho sa teda vývoj pre Android aplikácii silno odvíjal. Umožňuje podporu pri stavbe aplikácie, ponúka špecifické refaktorovanie – zmenu v kóde pre Android knižnice a rýchle opravy či návrhy. Podobne ako v iOS 3.3.1 aj Android má svoje nástroje na sledovanie výkonu či kompatibility a sú to nástroje Lint. Android



Obr. 4.2: Príklad vývojárskeho prostredí Android Studia (snímek obrazovky)

Studio ponúka tiež podporu na vývoj Android Wear a iných technológií ako Google Cloud Platform či Google App Engine.

4.4 Material Dizajn Android aplikací

V posledných rokoch bola najvýraznejšou zmenou v Androide zmena dizajnu. S príchodom Androidu 5.0 Lollipop priniesol Google absolútne nový dizajn s názvov Material dizajn. Pre dizajnérov navrhujúcim s princípmi materiálového dizajnu vyzdvihoval nasledujúce princípy [13]:

- *Materiál je metafora:* Jednotlivé časti aplikácie by sa mali chovať ako skutočné materiálne objekty.
- *Výrazný, grafický a zámerný:* Prechody medzi stránkami aplikácie by mala byť navrhnutá ako v kvalitnej knihe či magazíne.
- *Pohyb dodáva zmysel:* Aplikácia by mala adekvátne reagovať na používateľa animáciami.

4.5 Shrnutí Android vývoje

Najväčšou výhodou Android vývoja je veľká dostupnosť. Či už na strane potenciálnych používateľov aj na strane programátorov. Rovnako jazyk JAVA je vďaka open source - voľne prístupným zdrojom populárny a často krát preferovaný medzi nielen študentmi, ale aj profesormi a preto sa mu na školách venuje väčšia pozornosť.

Kapitola 5

Návrh aplikace

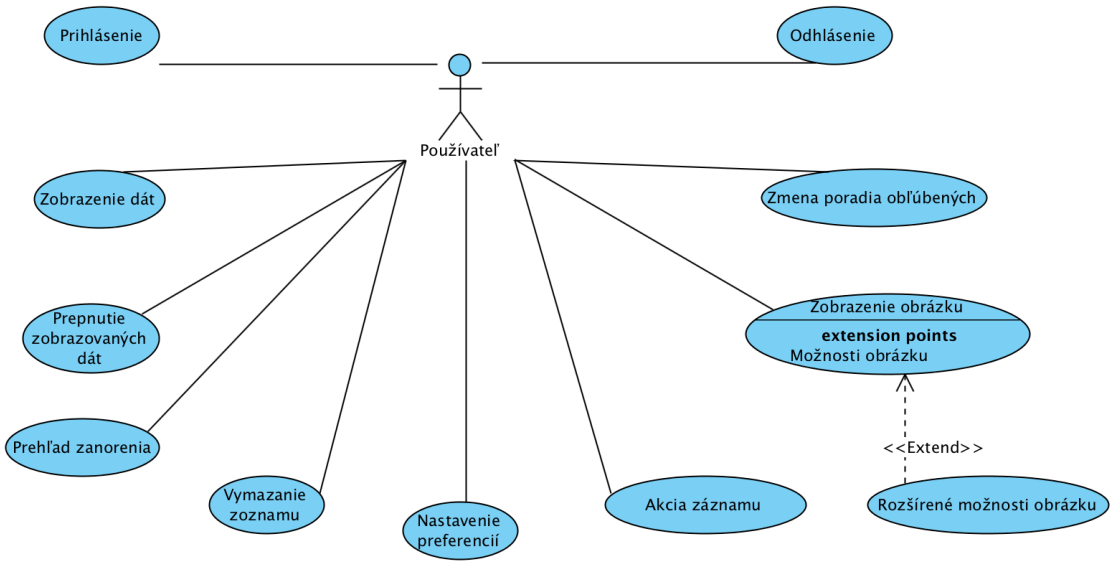
Nasledujúca kapitola sa venuje návrhu aplikácie. Aplikácia bude finálne distribuovaná pre obe zariadenia iOS ako aj Android no v tejto kapitole budeme rozoberať spoločné črty a návrh jadra systému. Výhodami aplikácie bude jej univerzálnosť a teda jednoduché nasadenie pre širokospektrálnych potenciálnych zákazníkov. Cieľom aplikácie je vytvoriť klient-server aplikáciu, kde klientská mobilná aplikácia bude (okrem domény) nezávislá od servera. Programátor serverovej časti bude teda absolútne ovládať systém mobilnej aplikácie, bez nutnosti zásahu do funkčnosti natívneho mobilného jadra. Výhodou je, že môžeme odbremeniť pri zmenách programátorov mobilných aplikácii a všetka logika je sústredená na jednom mieste. Cieľom je vytvoriť maximálne univerzálnu a nastaviteľnú aplikáciu s možnosťou prehľadu textov a súborov. Serverová časť bude mať možnosť nastavenia dizajnu aplikácie aj obsahu aplikácie.

5.1 Motivace

Typickou požiadavkou na výsledný systém je jednoduché zobrazenie dát z jedného alebo viacerých systémov a ich databáz. Serverová časť pomocou unifikovaného tvaru, konkrétne zvolený JSON, zašle dáta a zariadenie ich zobrazí. Používateľ pomocou zanárania v zozname posiela požiadavku na systém o nové dáta relevantné ku predošlým.

Logiku pre daný systém na strane servera určuje už konkrétny prípad, ale môžeme sledovať, že mobilná strana bude vykonávať vždy rovnakú funkcionalitu. Zobraz mi zaslané informácie, prípadne požiadaj o nové. Motiváciou projektu je teda vytvoriť unifikovanú aplikáciu pripravenú na využitie ľubovoľnou serverovou stranou na vlastné použitie. Pre každú jednu firmu, ktorá by potrebovala sprístupniť svoje dáta používateľom sa danou aplikáciou znížia náklady na developerov o 50-66 %, nakoľko je nutné vytvoriť logiku iba serverovej časti. Je preto veľké množstvo projektov, kde sa práca opakuje a stáva sa redundantnou. Výhodou je tiež dynamickosť vyvinutej aplikácie v zmysle zmeny podoby a rýchlosti nasadenia.

Princíp klient-server aplikácie nám dovoľuje meniť a prispôsobovať aplikáciu na vlastné použitie bez nutnosti zasahovania do mobilnej časti kódu a to umožnením veľkej flexibilnosti a nastavení zo strany serveru na obsah aj dizajn aplikácie. Typicky modifikácia aplikácie predstavuje veľa práce aj pri drobnej zmene, čo uvádzaný prístup eliminuje.



Obr. 5.1: Diagram použití uživatelské navržené aplikace (zdroj vlastní)

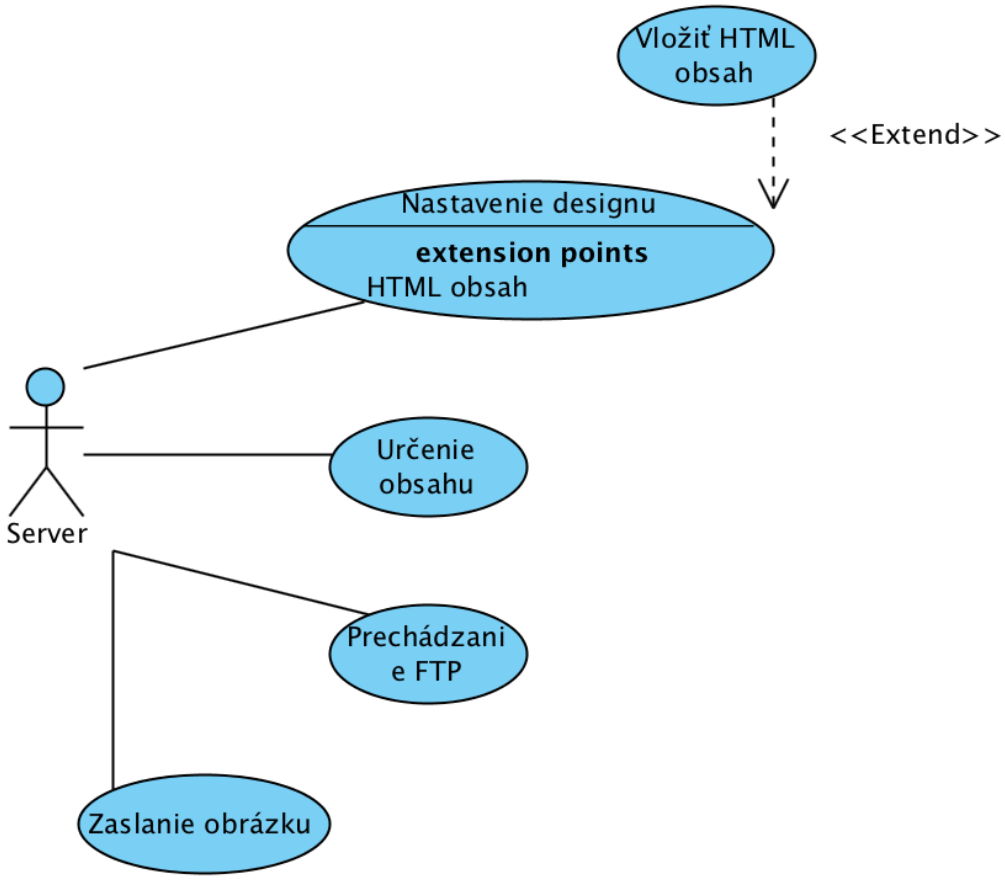
5.2 Požadavky na aplikaci

- zaslanie požiadavky o dáta na serveri pomocou API služby
- prijatie JSON dát a ich správne zobrazenie
- funkcionalita na základe triedy objektu
- ukladanie do medzipamäte podľa určeného času
- možnosť nastavenia dizajnu cez server
- zobrazenie dát rôznych formátov
- zanáranie dát
- možnosť označenia dát
- možnosť nastavenia poradia dát
- vyhľadávanie v dátach

5.3 Použití aplikace

V obrázku zobrazujúcom používateľský diagram použitia 5.1 môžeme sledovať, aké všeobecné možnosti bude aplikácia samotná poskytovať pre používateľa.

Môžeme tiež vidieť diagram prípadov použitia 5.2 pre samotného programátora serverovej časti a čo všetko so systémom môže robiť.



Obr. 5.2: Diagram použití serverové strany navržené aplikace (zdroj vlastní)

5.4 Architektura

V rámci návrhu bolo potrebné navrhnúť celkovú architektúru fungovania systému, ako aj architektúru mobilnej klientskej časti, ktorej sa budeme venovať.

5.4.1 Architektura celkového systému

Celková architektúra chodu systému je znázornená na obrázku 5.3. Môžeme sledovať dátové úložisko, server a mobilnú aplikáciu. Táto architektúra je veľmi všeobecná, pretože cieľom projektu je vyhovieť čo najväčšej a najtypickejšej problematike.

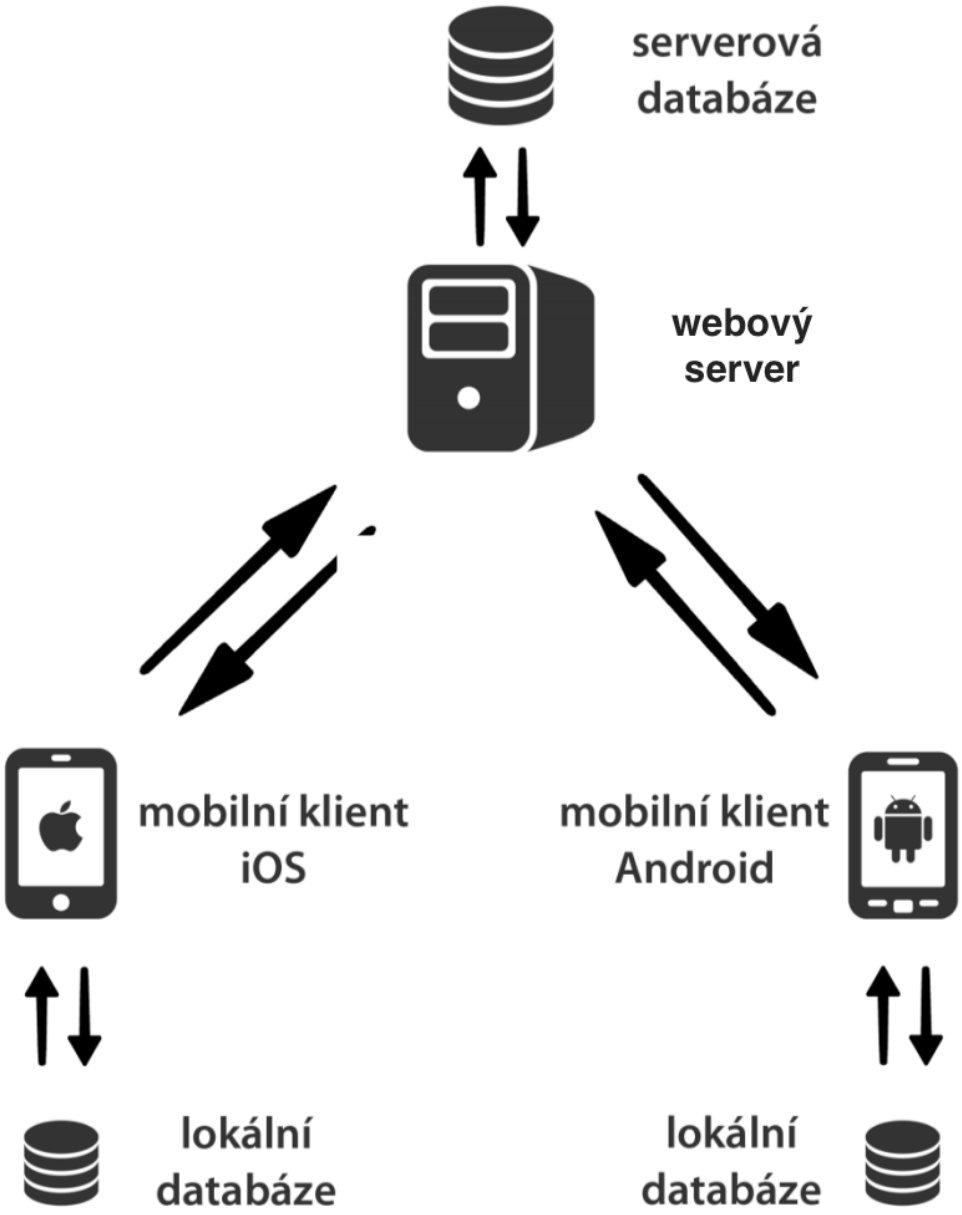
5.4.2 Architektura klientské mobilní aplikace

Projekt sa sústredí na mobilnú aplikáciu, ktorej cieľom je správne navrhnutie tak, aby bola efektívna, unifikovaná a nezávislá na serveri a zároveň vedel server dynamicky s aplikáciou pracovať a prispôsobovať si ju vlastným potrebám bez nutnosti zásahu do jej fungovania. Unifikovanosťou sa vyhneme komplikovanému nahrávaniu aplikácii na aplikačné servery či zásahu do kódu. Aplikácia bude vlastným logickým systémom, jadrom - engine - zobrazovania dát.

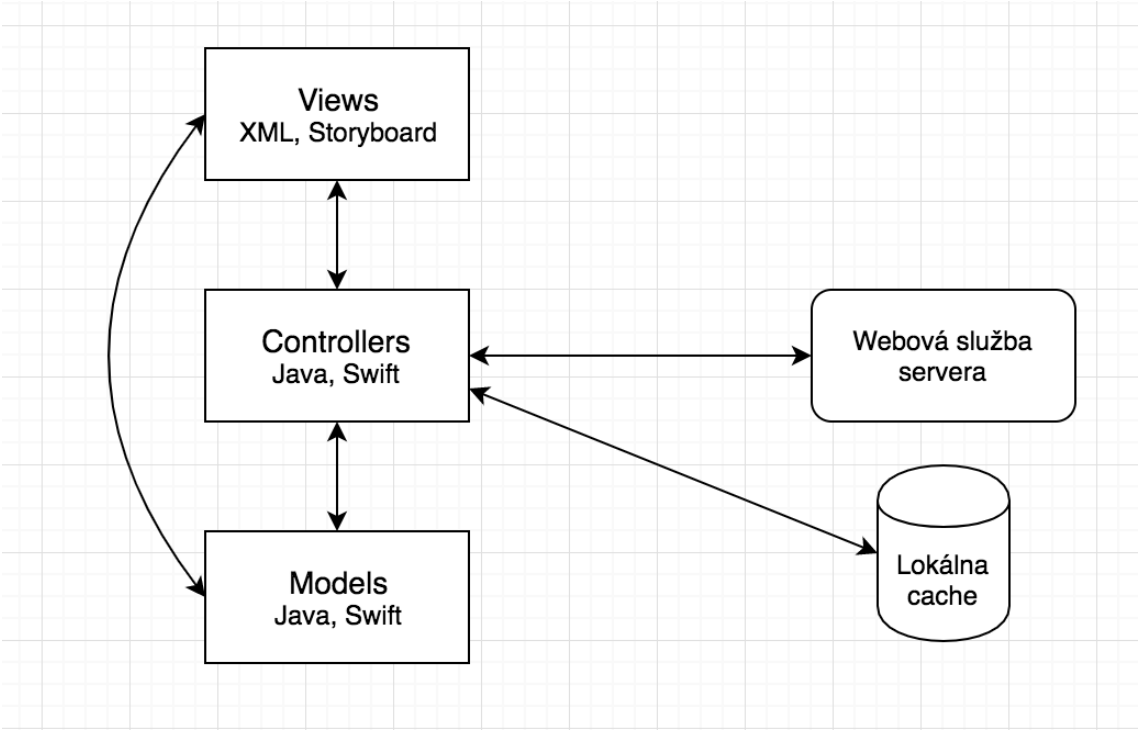
Jej architektúra je prirodzene vedená do architektúry Model-View-Controller 5.4.

Keďže požiadavky na aplikáciu nezahŕňajú off-line mód bez internetu, nemusíme sa zaoberať trvalým úložiskom v mobile, ale musíme vyriešiť ako súbory efektívne ukladať do medzipamäte – cache - počas používania aplikácie. Keďže dĺžka intervalu, prípadné trvalé stiahnutie pri dátach, ktoré sú nemenné, je nastavovaná serverom je proces spracovania dát rôzny 5.5.

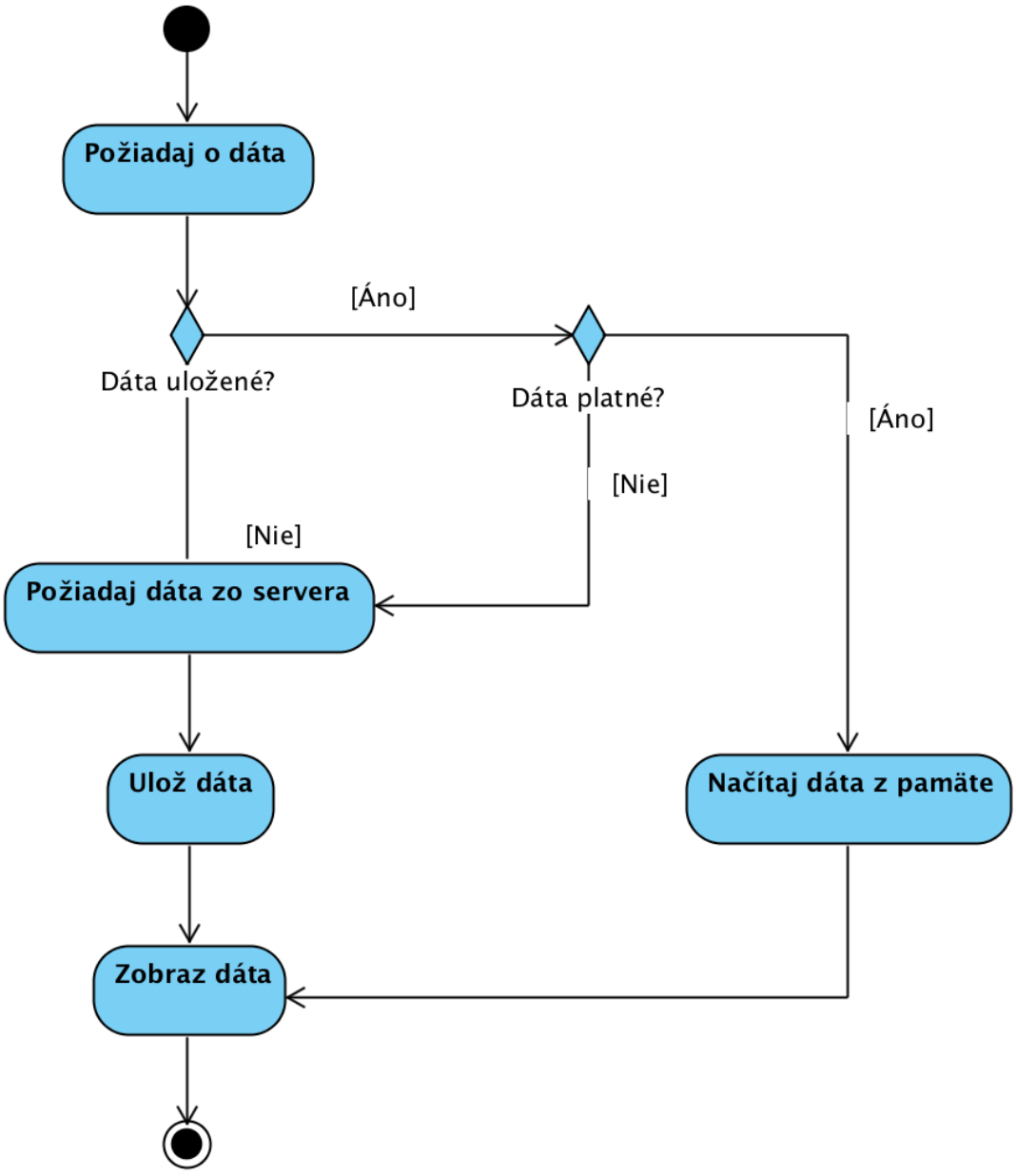
Hlavnými oporami projektu bude tabuľkové zobrazenie dát preberané v kapitolách iOS 3.2.6 aj Android 4.2.6 a taktiež webové preberanie JSON súborov a ich asynchrónne načítanie. Povaha aplikácie nám dovoľuje využiť všetky techniky, ktoré boli preberané v úvodných kapitolách. Využijem tiež ukladanie dát do medzipamäte - dočasného úložiska, aby bola aplikácia čo najefektívnejšia, no zároveň je serveru dovolené pre jednotlivé dátové sady danú medzipamäť nastaviť podľa vlastných potrieb, čím je opäť flexibilnejšia.



Obr. 5.3: Architektura celkového chodu aplikace (inspirováno [2])



Obr. 5.4: Architektúra klientské časti aplikacie (zdroj vlastní)



Obr. 5.5: Způsob přístupu ke získaným datům ze serveru - *mezipaměti(cache)* (zdroj vlastní)

Kapitola 6

Implementace

V nasledujúcej kapitole opíšem implementáciu navrhnutej aplikácie v prostredí iOS aj Androidu. Preberiem najdôležitejšie prvky vývojového cyklu aplikácie, podobnosti a rozdiely v rámci oboch prostredí. Zameriam sa na elementy používateľského rozhrania. V závere kapitoly rozoberiem najväčšie výhody aj nevýhody programovania z pohľadu implementácie.

6.1 Datový model a struktura aplikace

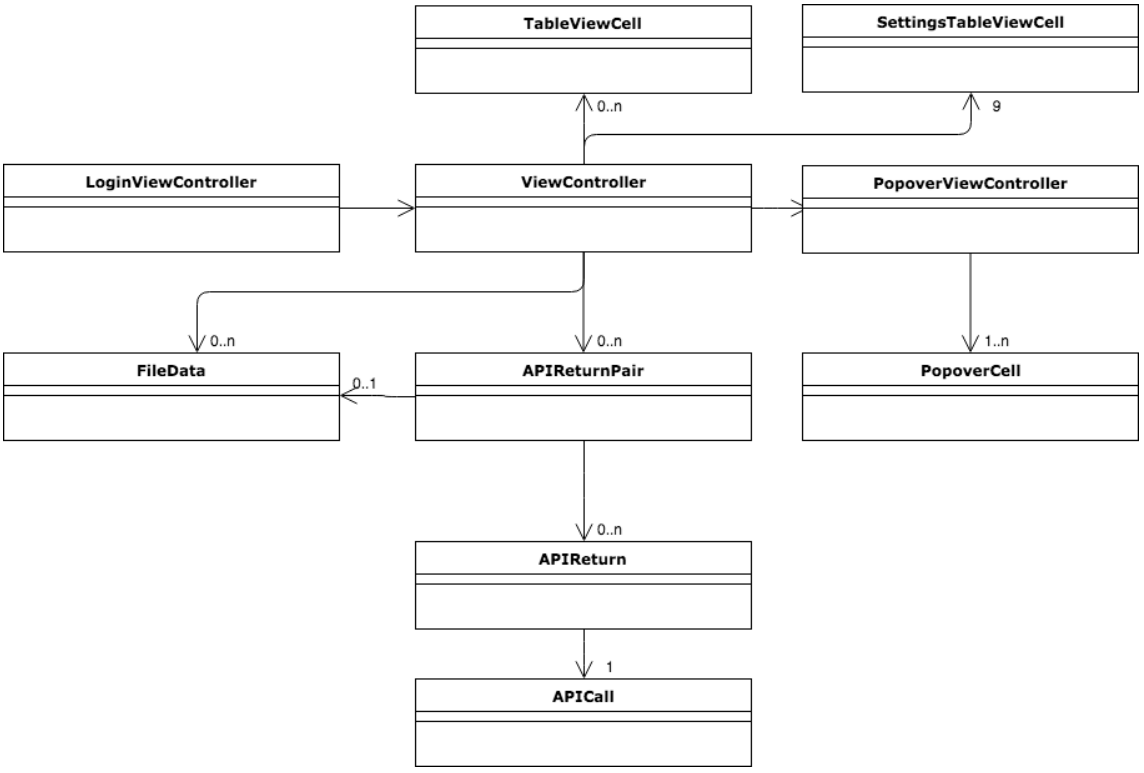
Dátový model aplikácie sa odvíja od štruktúry objektov webových služieb. Základ tvoria vstupné parametre webových služieb a následne ich dátová odpoveď. Základnými objektami sú:

- *trieda APICall*, obsahujúca štruktúru so základnými dátami vkladanými do webovej služby. Obsahuje nasledujúce charakteristiky:
  - *fn* - vkladaný názov volanej funkcie
  - *id* - id volaného zoznamu
  - *my_hash* - výpočet kontrolnej sumy haš zabezpečenia. Viac v kapitole zabezpečenie 6.2.6.
  - *nazov* - názov(text volaného `APIReturn`) volaného zoznamu
  - *typ* - typ volaného zoznamu
  - *trieda* - trieda volaného zoznamu
  - *hlbka* - hĺbka zanorenia volaného zoznamu
  - *zmz* - nastavenie označenia obľúbenosti daného prvku
  - *objem* - nastavenie preferencií používateľa na objem sťahovaných dát
  - *apiReturn* - objekt, z ktorého je služba volaná pre doplňujúce údaje
  - *usr* - trieda obsahujúca meno a heslo používateľa potrebné na vytvorenie haš funkcie
- *trieda APIReturn* obsahujúca samotné údaje jedného riadku tabuľky. Pomocou týchto údajov je aplikácia schopná zobraziť požadovanú informáciu, ale tiež sa hlbšie zanoriť v dátach a na server poslať po vnorení vhodnú požiadavku. Obsahuje nasledujúce charakteristiky:
  - *id* - id volaného zoznamu

  - *text* - text primárneho riadka prvku tabuľky
  - *text2* - text sekundárneho riadka prvku tabuľky
  - *typ* - typ volaného zoznamu
  - *kategoria* - sekcia, do ktorej je v tabuľke prvok zaradený
  - *icona* - názov ikony použitej v riadku tabuľky (číselná reprezentácia z balíku možných ikon, v prípade neznámej použitá prednastavená ikona)
  - *trieda* - trieda volaného zoznamu
  - *hlbka* - hĺbka zanorenia volaného zoznamu
  - *pocet* - počet prvkov, ktoré obsahuje vnorený zoznam daného prvku (v prípade -1 je číslo príliš veľké na zaslanie alebo nás počet nezaujíma)
  - *mark* - údaj, či je prvok zaradený medzi obľúbené položky
  - *styl* - maximálny počet riadkov pre primárny aj sekundárny text v riadku tabuľky, v prípade hodnoty 9 je riadok neobmedzený a jeho výška je nastavená podľa daného textu
  - *color* - jemné zvýraznenie farby riadku tabuľky
  - *html* - možnosť html formátovania textu tabuľky 6.3.4
  - *id0* - unikátne číslo pre vstup webovej služby *set_mark_all*, pre vytvorenie textového id poradia 6.3.6

- *trieda APIReturnList* obsahujúca základné údaje o dátovej sade, ktoré sú z navráteného JSON formátu ukladané. Obsahuje nasledujúce charakteristiky:
  - *title* - názov danej dátovej sady, určuje title v aplikácii aj v navigačnom strome
  - *pole objektov triedy APIReturn* - sada objektov definujúcich údaje tabuľky
  - *apiCall* - trieda, ktorá bola použitá na získanie aktuálneho zoznamu, použitá pri obnovení údajov, opätovnom volaní rovnakej služby
  - *offsetPair*[1] */ offset* - zachovanie posunu tabuľky, aby sa pri prechode späť v dátach používateľ nachádzal na rovnakom mieste
  - *callDate* - uchovanie informácie o čase zavolania dát pre budúce overenie platnosti dát
  - *autorefresh* - doba v sekundách po akej sú dáta opätovne obnovované v pravidelnom intervale 6.2.5
  - *cache* - doba v sekundách po akej dáta prestávajú byť platné. V prípade -1, sú dáta platné trvalo (okrem vynúteného obnovenia dát potiahnutím alebo stlačením tlačidla obnovenia) 6.2.2
  - *icona* - názov ikony použitej pre zoznam v navigačnom strome (číselná reprezentácia z balíku možných ikon, v prípade neznámej použitá prednastavená ikona)
  - *fileData* - v prípade dokumentov ukladáme dáta do objektu triedy FileData, ktorá obsahuje cestu k uloženému súboru ako aj ich typ formátovania
  - *iOS: jazyk* - jazyk aplikácie, využitý v iOS verzií
  - *iOS: vyska_auto* - zapnutie automatickej výšky riadkov, implementované, kvôli pomalému indexovaniu pri veľkom množstve dát v iOS verzií

[1] v Androide je potrebné si pamätať pozíciu aj posun preto sa ukladá offsetPair



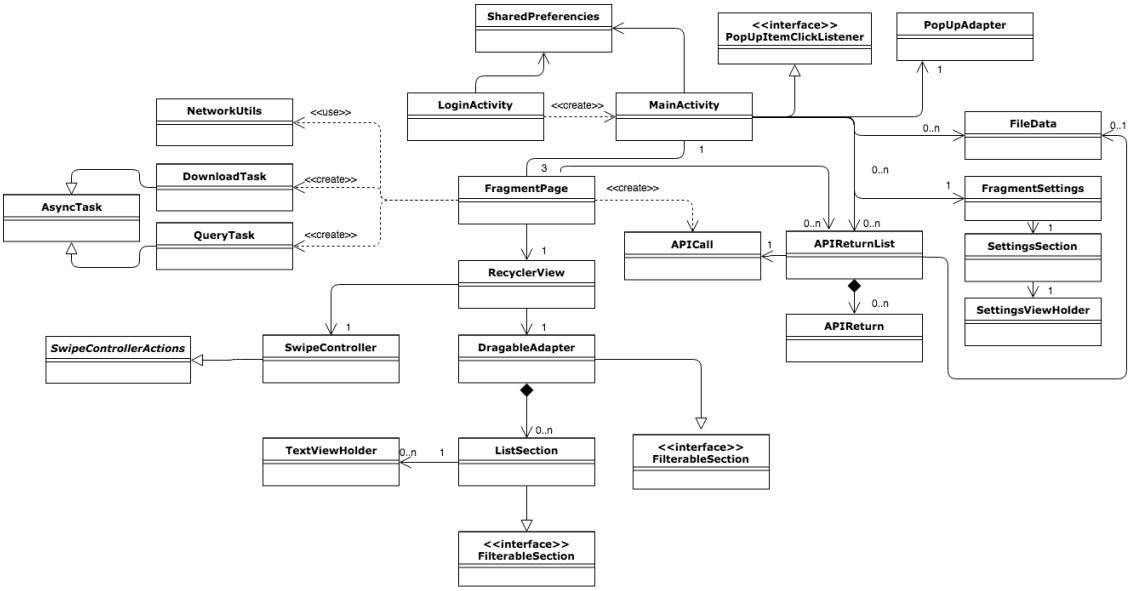
Obr. 6.1: Diagram tříd iOS aplikace (zdroj vlastní)

  - *iOS: rychly_vyber* - zapnutie možnosti zobrazenia pravého rýchleho presunu v tabuľke k vybranej sekcii. Index sekcii môže byť: 0 - vypnutý - iba jazdec, 1 - zapnutý s menami kategórii, 2 - zapnutý s očíslovanými sekciami
  - *iOS: hladanie* - zapnutie možnosti zobrazenia vyhľadávacieho extra tlačidla na hľadanie v tabuľke iOS

- *trieda User* trieda nesúca údaje o používateľovi ako meno a heslo
- *trieda FileData* trieda uchovávajúca miesto uloženia súboru a jeho typ internetového média(MIME typ)
- *trieda OffsetPair* trieda uchovávajúca pozíciu aj posun - offset danej tabuľky

6.1.1 Použita struktura aplikace iOS

V prípade iOS implementácie sa veľká väčšina logiky odohráva triede `ViewController`. Táto trieda deleguje funkcionalitu tabuľky, webového prehliadača aj vyhľadávania nad tabuľkou. Úvodné zobrazenie, kde sa používateľ prihlasuje zabezpečuje trieda `LoginViewController`. Jednotlivé obrazovky kontrolérov sa obsluhujú pomocou triedy `Segue`. Tieto konkrétne prepojenia obrazoviek sa navrhujú v súbore *Main.storyboard*. Následne sa pomocou vlastného *id* daný prechod zavolá a vykoná. Vložené údaje používateľa ako meno a heslo sa pred samotným vykonaním prechodu *segue* nastavujú vo funkcii `prepareForSegue`. V diagrame tried 6.1 môžeme sledovať nadväznosť tried.



Obr. 6.2: Diagram tříd Android aplikace (zdroj vlastní)

6.1.2 Použita struktura Android

V Android aplikácii je okrem hlavnej *aktivity* `MainActivity` vykonávajúcej hlavnú obsluhu, pomocnej aktivity `LoginActivity` v ktorej sa používateľ prihlasuje, využitá technológia *Fragmentov*. Fragmenty fungujú ako samostatná aktivita podriadená hlavnej aktivite. V aplikácii je implementovaná vnútorná trieda `Fragment_page` triedy `MainActivity`, ktorá obsluhuje tabuľku údajov pomocou externej knižnice *SectionedAdapter* a vlastnej triedy `ListSection`, asynchrónne volanie webovej služby pomocou triedy *AsyncTask* a z nej vlastnej triedy `QueryTask` a na ukladanie dokumentov `DownloadTask`. Rovnako sa vlastná trieda využila pre dosiahnutie posúvateľného riadka v tabuľke pre označenie a pre nastavenie poradia riadkov v sekcii Mark na dlhé podržanie je využitá trieda `ItemTouchHelper.Callback`. Celkový diagram tried môžeme sledovať v diagrame tried 6.2.

6.1.3 Porovnání datového modelu a struktury aplikace v iOS a Androidu

V oboch prípadoch je základná štruktúra aplikácie veľmi podobná. Úvodná obrazovka je implementovaná pomocou vlastnej aktivity, kontroléru a následne je hlavná logika v samostatnej aktivite, kontroléri. V prípade iOS sú jednotlivé tabuľky spravované aktivitou rovnako ako všetky webové prehliadače či spracovania dát. V prípade Androidu je logika spracovania tabuľky a volania webových služieb delegovaná do triedy Fragment a hlavná aktivita spravuje uložené dáta a úvodné nastavenia navigačnej aj akčnej lišty. Logiku Fragmentov možno použiť aj v iOS pomocou triedy `ContainerViewController`, no táto možnosť nie je implementovaná. Rovnako môžeme sledovať tendenciu vytvárať si vlastné triedy v Jave, kde naopak v iOS je možné funkcionalitu združovať v rámci jednej triedy `ViewController`. Tento nedostatok rieši čiastočne jazyk Kotlin 4.2.1, ktorý dovoľuje využívať funkcionalitu nadradených tried vo funkciách bez dedenia.

6.2 Spracování dat aplikace

Aplikácia samotná je univerzálnym nástrojom na prehľad textových a dokumentových dát zo servera. Dáta prichádzajú vo forme JSON súboru, predstavovaného triedou *APIReturnList* alebo ako dáta dokumentu ľubovoľného formátu 6.2.3 zobrazované vstavaným aplikačným webovým prehliadačom a ukladané na zariadenie. Využíva sa preto súbor univerzálnych API webových služieb poskytnutých serverom (implementácia serverovej strany, nie je rozborom tejto práce, nakoľko práca je zameraná na mobilnú aplikáciu, ktorá je schopná po zmene základu url pracovať s ľubovoľným spracovaním webových služieb). Po prihlásení do aplikácie(neúspešné prihlásenie je ďalej rozoberané v kapitole zabezpečenie 6.2.6) sú volané 3 základné, koreňové zoznamy pomocou funkcie *makeFirstQuery*. Volanie webových služieb je iniciované používateľovým stlačením riadku tabuľky - definovaného triedou APIReturn. Tieto dáta sú vkladané ako vstupné parametre funkcie *get_** Po získaní dát sa dáta uložia a vhodne zobrazia.

6.2.1 Použité webové služby

V aplikácii boli použité webové služby, volané na základe typu riadku tabuľky vybraného používateľom. Tieto služby sú:

- *get_zoznam* - úvodné volanie dát koreňových adresárov a volanie dát typu Text a Dir, návratom je APIReturnList
- *get_detail* - volanie dát typu Detail, návratom je APIReturnList
- *get_file* - volanie požadovaného súboru typu File, návratom sú dáta dokumentu
- *get_file2* - volanie požadovaného súboru typu File2, návratom sú dáta dokumentu
- *get_search* - volanie služby na vyhľadanie v dátach, návratom je APIReturnList
- *set_mark* - volanie služby na označenie obľúbenosti záznamu, hodnotami *id, trieda, hĺbka*, určujeme o aký prvok sa jedná a následne hodnotou *zmz*, vloženou do url určíme, či chceme riadok pridať alebo odobrať z obľúbených, návratom je číselná hodnota hovoriaca o úspešnosti vykonanej služby
- *set_mark_all* - volanie služby na určenie poradia, ktoré si používateľ zvolil - hlavným vstupom tejto služby je hodnota id, ktorá pozostáva zo zoradeného reťazca s hodnotou riadku *id0* a jeho aktuálnym poradím, vo forme *id0:poradie* spojené v jednom textovom reťazci pre všetky zoraďované prvky rozdelené čiarkou, návratom je číselná hodnota hovoriaca o úspešnosti vykonanej služby
- *del_cache* - služba umožňujúca vymazať zoznam s históriou alebo označenými položkami, vymazanie zoznamu na základe zaslanej triedy

Každý prvok tabuľky APIReturn určuje typ dát, ktorým je daný riadok hierarchicky nadradený. Na základe tohoto typu je vhodne volaná webová služba, prípadne vhodne vkladané vstupy do nej. Týmito typmi sú:

- *Text* - základný typ dát, volaná služba je *get_zoznam* a vstupné údaje do služby ako id, trieda, hĺbka sú preberané z nadriadeného, zvoleného objektu riadku tabuľky
- *Detail* - veľmi podobný typu Text no na jeho volanie využívame službu *get_detail*

- *File* - služba, ktorej vstupy sú opäť preberané zo zvoleného objektu tabuľky no odpoveď tejto služby je dokument v ľubovoľnom štandardnom formáte ako: jpg, png, pdf, docx a iné.
- *Dir* - typ dát, ktorý nám určuje adresárový spôsob práce s dátami. V typoch ako Text či Detail je id funkcie preberané z predchádzajúceho zvoleného objektu. V tomto prípade sa id tvorí ako nová cesta s predchádzajúceho id konkatenovaného - spájaného so štandardným znakom / a primárnym textom zvoleného objektu - simulujeme prechod dokumentovým stromom serverovej knižnice
- *File2* - rovnako ako v prípade typu *File* je návratovou hodnotou tohoto typu očakávaný súbor no opäť ako v prípade typu *Dir* je id volaných dát vytvárané a simuluje prechod adresárovým stromom
- *Search* - typ dát štrukturálne rovnaký návratovou hodnotou typu *Text* a *Dir* no svojim typom určujú, že dáta vznikajú volaním služby *get_search* a teda nad serverovými dátami je vyhľadávané vložené id nad aktuálnou triedou

Volání webové služby v iOS

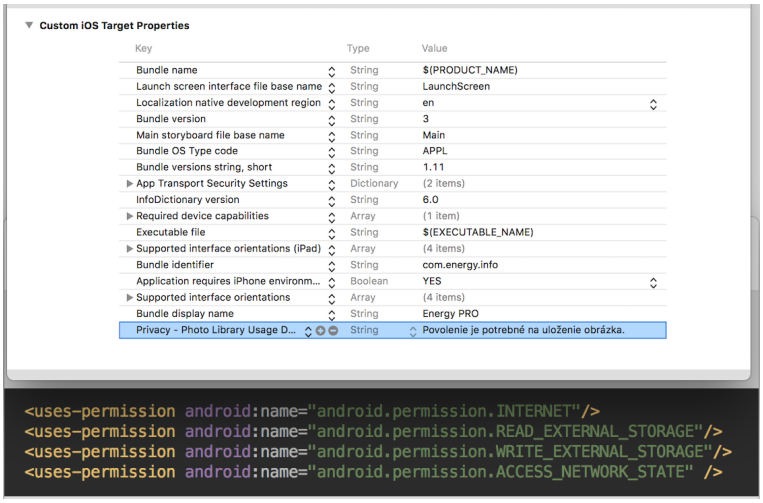
V iOS aplikácii je volanie služieb implementované v rámci hlavnej triedy *ViewController* vo funkcii *callAPIPost*. Táto funkcia je zodpovedná za vytvorenie vhodnej URL zo vstupov, oslovenie webovej služby pomocou triedy `URLRequest(url: url)`, ktorá je vstupom do asynchrónneho volania triedy `URLSession.shared.dataTask(with: request)`. Funkcia je zodpovedná aj za následné spracovanie navrátených dát z JSON formátu do vlastných tried `APIReturnList` pomocou triedy `JSONSerialization.jsonObject`. Pomocou closures 3.2.5 sa na ukončenie funkcie upravenia potrebné objekty používateľského rozhrania.

Volání webové služby v Androide

V Android verzii aplikácie sa volanie webovej služby vykonáva pomocou vlastnej triedy DowloadTask zdedenej z triedy `AsyncTask`. Táto triedu sa využíva na stiahnutie dát na pozadí, prijatie dát, uloženie JSON súboru do vlastných štruktúr pomocou triedy `JSONObject` a následné upravenie používateľského rozhrania UI. Funkcionalita vytvárania vhodných štruktúr na zasielanie a prijímanie webových služieb implementuje vlastná triedu `NetworkUtils`. Táto trieda poskytuje funkcie na vytvorenie vhodnej `URL` triedy pomocou triedy `Uri.Builder`, ktorá je potrebná pri volaní webových služieb. Rovnako trieda NetworkUtils implementuje funkciu `getResponseFromHttpUrl(URL url)`, ktorá oslovuje vloženú url adresu pomocou triedy `HttpURLConnection` a spracováva odpoveď pomocou tried `InputStream` a `Scanner`.

6.2.2 Způsob ukládání souborů do mezipaměti

Aplikácia si priebežne pri prechode tabuľkou a práci s webovými službami dáta ukladá. Kontrolu ich platnosti, ich načítanie z pamäte či opätovné stiahnutie iniciuje vo funkcii `loadOrDownloadData` na základe požadovanej dĺžky *medzipamäte* v triede APIReturnList. Každý prijatý objekt triedy APIReturnList je uložený v hlavnej `MainActivity` pomocou triedy `HashMap<String, APIReturnList>` v prípade Androidu a pomocou triedy `ViewController` do štruktúry `NSMutableDictionary` v prípade iOS. Dáta sú ukladané



Obr. 6.3: Způsob nastavení povolení aplikace (snímek obrazovky, vlastní)

pod jedinečným kľúčom vytvoreným z daného APIReturnList objektu pomocou jeho charakteristík, kde *kľúč = trieda + id + hlbka + fn + nazov*. Následne sa pri opätovnom volaní služby kontroluje, či požadované dáta volania už nie sú uložené a pokiaľ áno, či sú dáta aktuálne platné. Logický proces je znázornení na diagrame aktivít 5.5. Môžeme pozorovať, že prístup je v oboch prípadoch rovnaký.

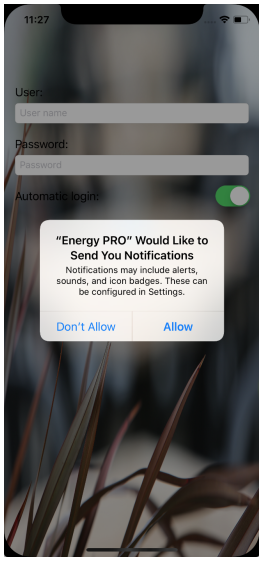
6.2.3 Práce se soubory

V aplikácii je možnosť prehliadania súborov zaslaných serverom a následná možnosť ich ďalšieho spracovania dostupnými aplikáciami v mobile. Pre zobrazenie štandardných obrázkových súborov ako jpg, png, ale aj atypických internetových typov mediálnych súborov ako: pdf, docx a iné je využitý komponent webového prehliadača. Všetka logika načítavania súboru je vo funkcii `loadFile`. Popri načítaní dát do webového prehliadača je rovnako na pozadí súbor ukladaný do zariadenia. Aby sa súbor mohol uložiť, je v oboch prípadoch potrebné, aby používateľ súhlasil s ukladaním dát na zariadení. Toto potvrdenie implementujú zariadenia samé na základe vložených požadovaných povolení. Ich nastavenie v oboch platformách môžeme sledovať na obrázku 6.3. V iOS je nastavovaný súbor s povoleniami *Info.plist* v Androide *AndroidManifest.xml*.

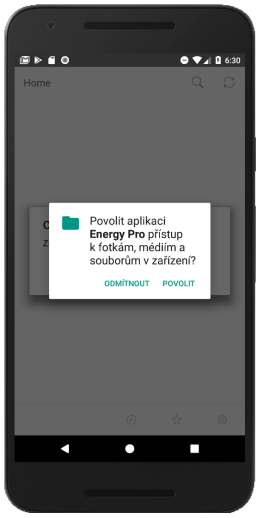
Na základe nastavení je používateľ vyzvaný k odsúhlaseniu požiadaviek. Môžeme vidieť na obrázku 6.4, že v prípade iOS, je text povolenia prispôsobený požiadavkám a používateľovi je dôvod povolenia vysvetlený, naopak v Androide 6.5 je text nastavený. Pri opätovnom načítavaní súboru je súbor načítavaný z pamäte. Môžeme pozorovať, že prístup je v oboch prípadoch principiálne rovnaký, no obmedzené sú možnosti komponenty webového prehliadača. Aby sa súbor mohol ďalej spracovávať, je potrebné, aby používateľ umožnil ukladanie súborov na zariadenie. Pokiaľ toto nastavenie povolil, je schopný súbor exportovať do externých aplikácii pomocou vlastnej funkcie `shareFile`.

Práce se soubory v iOS

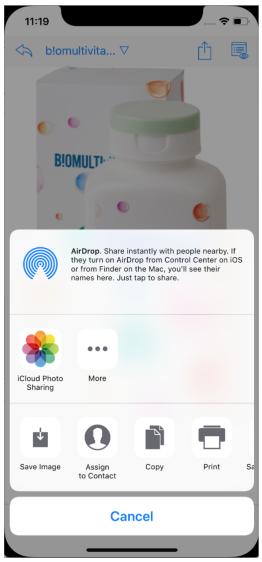
iOS verzia aplikácie poskytuje prehliadanie cez webový prehliadač schopný načítať aj špeciálne typy internetových dokumentových médií. Pomocou delegovania UIWebViewDelegate sú akcie webového prehliadača obsluhované v hlavnom `ViewController`. Aby mohol pou-



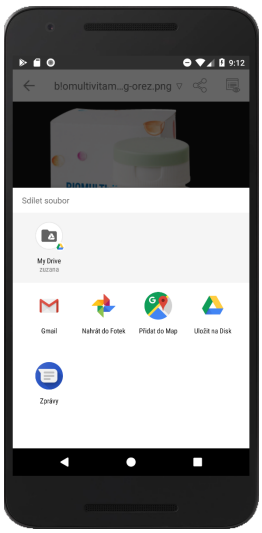
Obr. 6.4: Povolení ukládání dat iOS (snímek obrazovky, vlastní)



Obr. 6.5: Povolení ukládání dat Android (snímek obrazovky, vlastní)



Obr. 6.6: Export dokumentu iOS (snímek obrazovky, vlastní)



Obr. 6.7: Export dokumentu Android (snímek obrazovky, vlastní)

živateľ súbor delegovať ďalším aplikáciám, je potrebné ho uložiť. Miesto uloženia spravuje trieda `FileManager`. Získané dáta sa triedou `NSURLConnection` ukladajú pomocou funkcie `write` na vytvorenú url. Do webového prehliadača je následne možné načítavať už uložené dáta vložením aktuálnej cesty a typu média, čím dosahujeme efektívnejšiu prácu so súbormi. Pre možnosť exportu súboru je uložená cesta nastavená triede `UIDocumentInteractionController`, vďaka čomu má následne používateľ možnosť súbor otvoriť v externých aplikáciách pomocou funkcie `presentOptionsMenu`. Táto funkcia je volaná, pokiaľ používateľ stlačí tlačidlo v hornej lište vyhradené na export. Export môžeme sledovať na obrázku 6.6.

Práce se soubory v Androide

V aplikácii bolo potrebné sa zvlášť venovať špeciálnym typom dokumentových internetových médií. Tieto typy napríklad sú: *pdf, doc, docx, xls, ppt, tif*. Prípadne všetky iné neštandardné, iné ako základné webové *html, css, js, svg* a obrazové *jpg, png*. Pre načítanie uvedených typov súborov je využitá webová služba od Google *viewer*, kde sa vkladá parameter embeded=*true* a url s daným dokumentom. S takouto URL adresou je v Androide webový prehliadač schopný zobraziť aj špeciálne typy súborov. V aplikácii je preto implementované iba načítavanie z pamäte jednoduchých obrázkov ako jpg a png formát.
Možnosť exportu súboru poskytuje trieda `ShareCompat.IntentBuilder` a následne trieda `Intent`. Do tejto triedy je vložená uložená cesta súboru ako aj jeho typ. Pomocou funkcie `createChooser` sa zobrazí panel možných aplikácii na stlačenie tlačidla exportu. Výsledné zobrazenie v aplikácii môžeme sledovať na obrázku 6.7.

6.2.4 Preference uživatele

Preferencie používateľa ako uloženie mena a hesla, úvodná stránka pri načítavaní aplikácie, automatické prihlasovanie a dopyt po objeme dát sa ukladá do perzistentného úložiska, ktoré ponúkajú obe platformy. V oboch prípadoch sa jedná o slovník v ktorom máme hodnoty uložené pod textovým kľúčom.

Perzistentné úložisko v iOS

iOS ponúka triedu `UserDefaults.standard`, pre ktorú je možné registrovať slovník definovaných preferencii. Slovník preddefinovaných a dostupných hodnôt sa iniciuje v triede `AppDelegate` vo funkcii `didFinishLaunchingWithOptions` a následne je možné s hodnotami a triedou pracovať.

Perzistentné úložisko v Androide

Android ponúka triedu `SharedPreferencies` na získanie aktuálne nastavených preferencií aplikácie. V prípade, že sa prvok v danom zozname nenachádza vráti sa prednastavená hodnota - jedna zo vstupov funkcie. Na editáciu sa využíva `SharedPreferencies.Editor`, kde po zmene dát je vždy nutné zavolať funkciu `apply`

6.2.5 Automatická obnova dat

V aplikácii je implementovaná automatická obnova dát v určitom intervale. Táto hodnota je daná serverom charakteristikou *autorefresh* v objekte APIReturnList. Po uplynutí dohodnutej doby sú dáta aktuálnej tabuľky obnovené. V oboch prípadoch je implementovaný časovač, ktorý v určenom intervale zavolá poslednú zavolanú službu, s dátami uloženého objektu APICall.

Implementovaný časovač v iOS

iOS Swift ponúka možnosť použitia triedy *Timer* (v skorších verziách Swift jazyka NSTimer) a jeho funkciu `scheduledTimer`. V tejto triede je pomocou triedy Selector a vlastnej funkcie `action_refreshTimer` vďaka forme *#selector* implementované obnovovanie dát aktuálnej tabuľky.

Implementovaný časovač v Androide

Android ponúka viacero možností, pomocou ktorých je možné implementovať časovač[3] - funkciu, ktorá bude opakovane spustená v časovom intervale. Triedy, pomocou ktorých je možné implementovať časovač sú nasledovné:

- *Handler a Runnable*
- *Handler a Callback*
- *TimerTask a Runnable*
- *Timer*

V projekte som využila možnosť triedy *Handler* a jeho funkcie *postDelayed(Runnable r, int delayMillis)*. Vo vlastnom objekte triedy *Runnable* a vo funkcii *run* je vykonaná požadovaná funkciu časovača - obnovenie dát - a následne opätovne volaná funkcia *postDelayed*. V prípade, že je automatické nastavenie aktuálneho zoznamu vypnuté zo strany servera (nastavením charakteristiky *autorefresh*), je zavolaná funkcia triedy *Handler removeCallbacks*.

6.2.6 Zabezpečení

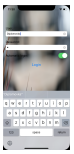
Používateľ je v úvode aplikácie vyzvaný k vloženiu mena a hesla. Zabezpečenie aplikácie pozostáva na dohode medzi serverom a klientom o zasielaní vypočítanej haš funkcie na základe používateľových vložených údajov. Server samotný je zodpovedný za registráciu povolených používateľov a ich následné overovanie. Aplikácia v každom volaní webovej služby vkladá vstup do url vygenerovanej *haš*, ktorá je následne overovaná s hodnotou na serverovej strane. Výpočet hodnoty haš sa vytvára funkciou md5[14], ktorej vstupom je požadované *id* a *trieda*, následne *meno* a *heslo* používateľa a aktuálny deň konkatenovaný do jedného textového reťazca. Úspešná odpoveď webovej služby je podmienená správnou haš funkciou, čím zabezpečujeme autentizáciu. Pre správny používateľský zážitok, (UX - user experience) je implementovaná možnosť automatického prihlasovania, prípadne pri odhlásení zapamätanie si mena a hesla.

6.2.7 Porovnáni spracováni dat v iOS a Androidu

Zásadným rozdielom, v ktorom má iOS výhodu oproti Androidu je, že po otočení zariadenia sa v Androide daná aktivita ukončí, stratí všetky aktuálne nastavenia a dáta a po otočení sa štartuje nová. Preto je programátor nútený všetky aktuálne dáta uchovať a následne po otočení obnoviť. V iOS nie je potreba niečo takéto riešiť, aplikácia aj dáta sa automaticky prenesú. Rovnako je zásadným rozdielom funkcionalita použitej webovej komponenty. Android neumožňuje využiť webový prehliadač pre univerzálne zobrazenie dokumentov iných ako typické obrazové formáty ako jpg a png. Toto obmedzenie vieme nahradiť zobrazením pomocou webovej služby Google, ktorej výstup je prehliadač schopný zobraziť. Bohužiaľ, kvôli tomuto nedostatku sa načítavajú súbory z pamäte iba pre dostupné formáty.

6.3 Uživatelské rozhraní aplikace

V prípade klient-server aplikácie som sa zamerala na používateľské prostredie a jeho možnosti. Implementovala som prácu s tromi tabuľkami líšiacimi sa triedou koreňového volania -



Obr. 6.8: Login (snímek obrazovky, vlastní)

Obr. 6.9: Tabulka (snímek obrazovky, vlastní)

Obr. 6.10: Dokument (snímek obrazovky, vlastní)

Obr. 6.11: Nastavení (snímek obrazovky, vlastní)

hlavným prehľadom, históriou, označenými prvkami. Ďalej je implementovaná práca so súbormi, zanáranie v hierarchii, vrchná lišta možností, navigačný strom na rýchli prechod v zanorení, kategórie dát, exportovanie súborov a iné možnosti rozoberané ďalej v nasledujúcej kapitole. Základné obrazovky môžeme sledovať na obrázkoch 6.8, 6.9, 6.10 a 6.11.

6.3.1 Implementovaný tabulkový přehled dat

Dáta získané webovými službami sú reprezentované tabuľkou, ktorá reaguje na stlačenie konkrétneho riadku, riadky zobrazujú prijaté dáta a prispôsobujú svoj vzhľad na základe typu, štýlu, či ikony. Ako bolo rozoberané v kapitole o tabuľkovom prehľade dát o iOS 3.2.6 a tabuľkovom prehľade dát v Androide 4.2.6 obe prostredia poskytujú pre danú problematiku komponent tabuľky, ich konkrétnu implementáciu rozoberieme v nasledujúcej kapitole.

Implementovaný tabulkový přehled dat v iOS

V iOS verzií je majoritná funkcionalita definovaná v hlavnom `ViewController` nakoľko okrem potomka triedy `UIViewController` implementuje tiež protokoly `UITableViewDelegate`, `UITableViewDataSource`. Delegované protokoly umožňujú implementovať všetky potrebné funkcie na prácu s tabuľkou ako určenie jej vzhľadu, dát a používateľove interakcie s ňou ako stlačenie riadku tabuľky. Pre vytvorenie vlastného vzhľadu riadku tabuľky bola vytvorená trieda `TableViewCell`. V tejto triede sú definované prvky riadku ako napríklad `UIImage` pre ikonu, `UILabel` pre text. Implementované sú aj pomocné funkcie na zmeny v rozložení objektov pomocou *constraints*.

Obnova dat potáhnutím tabulky v iOS Tabuľka ponúka možnosť obnovenia dát po potiahnutí dole na vrchnej pozícii, tak ako sú používatelia zvyknutí s natívnych či iných aplikácii mobilu. Sledovanie takéhoto gesta aj samotnú animáciu umožňuje trieda `UIRefreshControl`. Tejto triede je pri detekcii gesta definovaná vlastná funkcia `refreshItems` pomocou vstavanej funkcie `addTarget`. Následne je tento objekt nastavený pre požadovanú tabuľku pomocou funkcie `addSubview`. iOS navyše od Androidu ponúka možnosť nastavenia vlastného textu, zobrazeného pod tabuľkou po potiahnutí tabuľky smerom dole

na obnovenie. Do tohoto textu je pridávaná informácia o poslednej aktualizácii a počte prvkov v tabuľke pomocou atribútu *attributedTitle* triedy UIRefreshControl. Táto možnosť v Androide chýba.

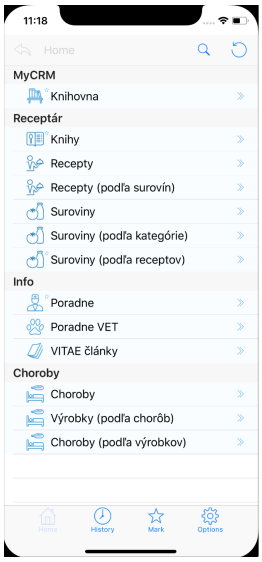
Implementovaný tabulkový přehled dat v Android

Každá trieda `FragmentPage` má svoj komponent `RecyclerView`, pre ktorú definuje vlastnú triedu `DragableAdapter`. Aby sa mohli v Androide zobraziť jednotlivé sekcie, využila som externú knižnicu *SectionedRecyclerViewAdapter*, z ktorej trieda adaptéru DragableAdapter dedí funkcie na vytvorenie jednotlivých sekcií. Funkcionalitu adaptéru ako vytvorenie potrebného počtu objektov riadkov a ich naplnenie definuje vlastná trieda `ListSection` zdedená z triedy `StatelessSection`. Táto trieda je spolu s pomocným rozhraním na filtrovanie v danej sekcii zodpovedná za spracovanie dát a za vytvorenie a naplnenie potrebného počtu objektov riadkov tabuľky vlastnou triedou `TextViewHolder`.

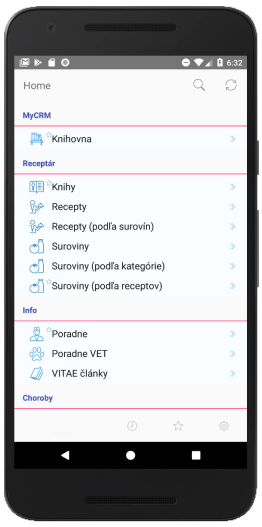
Obnova dat potáhnutím v Android Funkciu obnovenia dát tabuľky na potiahnutie dole implementuje komponent `SwipeRefreshLayout`. V zdrojových xml súboroch rozloženia fragmentu sa tento komponent implementuje ako vonkajší komponent, obaľuje teda komponent RecyclerView. Nie je preto nutné v programe tieto objekty prepájať. Funkciu obnovenia dát následne implementujeme pomocou objektu sledujúceho dané gestá `SwipeRefreshLayout.OnRefreshListener`.

6.3.2 Kategorie dat

Zasielané dáta obsahujú svoju kategóriu, podľa ktorej sú umiestnené v sekciách v tabuľke. Logicky združujú rovnaké dáta a robia tabuľku prehľadnou. Je potrebné dáta roztriediť a následne vhodne vložiť do sekcií. Ako sa funkcionalita implementovala v jednotlivých prostrediach rozoberiem v nasledujúcej časti.



Obr. 6.12: Základní obrazovka iOS (snímek obrazovky, vlastní)



Obr. 6.13: Základní obrazovka Android (snímek obrazovky, vlastní)

Kategórie dat v iOS

Sekcie v iOS sú prirodzene implementované v rámci funkcií protokolu `UITableViewDelegate`. Vzhľad sekcie je predurčený prostredím, čím je zachovávaný štandard iOS aplikácii. Konkrétny počet sekcií, počet riadkov v danej sekcii a mená hlavičiek sú implementované v rámci `ViewController` triedy, pomocou funkcií `numberOfSections`, `numberOfRowsInSection`, `titleForHeaderInSection`. Na udržiavanie roztriedených dát v rámci sekcií bola implementovaná pomocná štruktúra `DictTuple`, ktorá udržiava pole kategórií a následne pre každú kategóriu zoznam dát. Hlavičky iOS automaticky držia pozíciu na vrchnej časti obrazovky počas prechodu tabuľkou, aby používateľ vedel v akej kategórii sa nachádza. Android túto funkciu natívne nemá, mohla by sa použiť napríklad externá knižnica *StickyHeaders* [6], implementovaná v projekte, ale nie je. Výsledné zobrazenie môžeme sledovať na obrázku 6.12.

Kategórie dat v Androide

Sekcie tabuľky pre rozdelenie dát do kategórii v Androide prirodzene implementované nie sú. Pri implementácii je preto použitá externá knižnica *SectionedRecyclerViewAdapter* [12]. Ponúka funkciu adaptéra `addSection` do ktorej vkladám vlastnú implementovanú triedu `ListSection` zdedenú z triedy `StatelessSection`. Trieda ListSection sa chová podobne ako adaptér, definuje jednotlivé riadky danej sekcie pomocou funkcie `onBindItemViewHolder`, no ponúka navyše funkciu `onBindHeaderViewHolder`, ktorá určuje a napĺňa vzhľad hlavičky danej sekcie. V prípade Androidu, na rozdiel od iOS, je využitý vlastný dizajn pre hlavičku sekcie s farbami zvoleného štýlu používateľa 6.13.

6.3.3 Rychlý přesun tabulkou dat

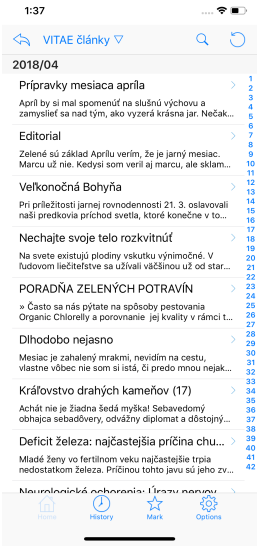
Pri veľkom množstve záznamov tabuľky je potreba používateľovi umožniť rýchli presun tabuľkou do stredu či na koniec. V oboch prostrediach sú možnosti implementovania rôzne. V prípade iOS je možné nastaviť na pravej strane tabuľky textové skratky pre presun na jednotlivé sekcie. Pokiaľ je táto možnosť vypnutá prirodzene sa na pravej vykresľuje veľkosť aj pozícia jazdca no nie je možné ním manipulovať, čo je nevýhodou v prípade, že máme tabuľku s jednou sekciou príliš dlhú. V prípade Androidu je implementovaný jazdec, s ktorým je možné sa v tabuľke pohybovať. Možnosť bočného indexu sekcií v Androide nie je. Rozdiel v dizajne oboch prístupov rozoberiem v tejto podsekcii.

Rýchly přesun tabulkou dat v iOS

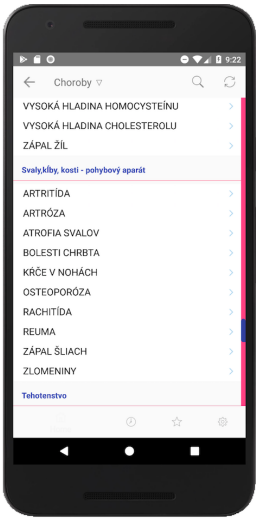
Presun na určené miesto v tabuľke je implementované pomocou natívnej funkcie `sectionIndexTitles` v ktorej je definované pole textových hodnôt, skratiek na presun v tabuľke. Zo serverovej strany je možné nastaviť pomocou hodnoty *rychly_vyber*, či je daný index viditeľný. Pomocou návratovej hodnoty `nil` je index automaticky vypnutý. Možnosť indexovania číselne je možné vidieť na obrázku 6.14

Rýchly přesun tabulkou dat v Androide

Pre vytvorenie posuvného jazdca Android ponúka možnosť nastavenia hodnoty *fastScrollEnabled* v komponente RecyclerView. Aby bol systém schopný vykresliť jazdca, bolo potrebné nastaviť zdrojové súbory `res/drawable` pre vykreslenie posúvacej plochy nazvanej *Track* a jazdca nazvaného *Thumb* pre vertikálny aj horizontálny smer. Tieto súbory som vytvorila



Obr. 6.14: Rychlá navigace v tabulce iOS (snímek obrazovky, vlastní)



Obr. 6.15: Rychlá navigace v tabulce Android (snímek obrazovky, vlastní)

pomocou komponent `selector, item` a `shape`. Pri výbere som využívala primárne nastavené farby štýlu používateľa pomocou kolekcie farieb `@color` poskytujúcej tri možnosti farby a to primárnu, primárnu tmavú a doplnkovú. Výsledok môžeme sledovať na obrázku 6.15. Bolo by možné použiť externú knižnicu *FastScroll* [10] na vykreslenie indexov pre rýchlu navigáciu. V projekte táto knižnica implementovaná nie je pre jej nedostatočnú flexibilitu.

6.3.4 Nastavení stylizace řádku

Nastavenie štylizácie riadku tabuľky je v prípade oboch prostredí veľmi podobné. Je potrebné pracovať s komponentom na prehliadanie textu, UILabel v iOS, TextView v Androide, a ich maximálnym počtom riadkov. Túto hodnotu je možné serverom nastaviť hodnotou *styl* triedy APIReturn. Hodnota *styl* je dvojciferné číslo, kde desiatková hodnota udáva počet riadkov v primárnom, hornom texte a jednotková hodnota počet maximálnych riadkov v spodnom texte. Rovnako sa nastavoval obrázok ikony, ktorej meno bolo určené zo strany servera, v prípade neznámej ikony je použitá prednastavená ikona. Zaujímavosťou je, že Android musí mať pomenované obrázky vo svojom zdrojovom adresáry res/drawable s prvým textovým znakom. V iOS obmedzenie na názvoslovie nie je. Upravujú sa tiež vzájomné umiestnenia jednotlivých komponent riadku tabuľky podľa *typu* objektu `APIReturn`, prípadne *počtu* nasledujúcich hodnôt. Pre typ *Text* sa zobrazuje navyše textové pole v krúžku, dosiahnuté obrázkom s rovnakou pozíciou ako text. Pre nulový počet sa ikona vpravo nezobrazuje, aby sa naznačilo, že po stlačení tlačidla nenastane prechod. Pre rôzne typy objektov sa následne nastavuje pravá ikona na zodpovedajúcu k typu ako ikona:

- *lupa* pre typ vyhľadávania *Search*
- *spinka* pre typ súboru *File,File2*
- *dvojitá šípka* pre zoznam *Text,Dir*
- *jednoduchá šípka* pre detail *Detail.*

Rovnako je možné serverom zapnúť HTML formátovanie textu riadku pomocou hodnoty `html` triedy `APIReturn`. Čím sa dosahuje flexibilnejšia práca s textom a vzhľadom.

Obr. 6.16: [caption]Nastavení html formátovaní textu řádku iOS (snímek obrazovky, vlastní)

Obr. 6.17: [caption]Nastavení html formátovaní textu řádku Android (snímek obrazovky, vlastní)

Nastavení stylizace řádku v iOS

V iOS je možné rozloženie komponent riadka tabuľky definovať v hlavnej triede rozloženia UI objektov používateľského rozhrania `Main.storyboard`. V programe sa vzhľad buniek dynamicky upravuje vo funkcii `cellForRowAt`, ktorá určuje vzhľad aj konkrétne rozloženie a triedu `SettingsTableViewCell` pomocou funkcie `dequeueReusableCell`. Vzhľad je upravovaný pomocou odkazov na *constraints* 3.2.3.

HTML stylizace textu v iOS Pre rýchly prístup a plynulé používanie HTML formátovania riadka tabuľky bola trieda `APIReturn` rozšírená o objekty tried `NSAttributedString` s nastaveným atribútom triedy `NSHTMLTextDocumentType`. Trieda `UILabel` vie následne tento atribútový textový reťazec vhodne zobraziť. Realizáciu v aplikácii môžeme sledovať na obrázku 6.16

Nastavení stylizace řádku v Androidu

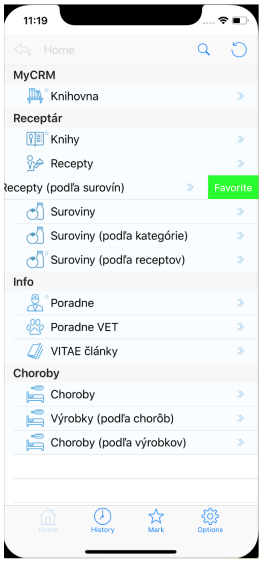
V Androide vlastnú zostavu a rozloženie komponent používateľského prostredia definuje, nový typ zdrojového res/layout súboru `rv_list_tablerow.xml`. Jednotlivé časti riadku v tabuľke následne predstavuje trieda `TextViewHolder`. Konkrétne hodnoty a vzhľad riadku nastavujeme vo funkcii `onBindItemViewHolder`. Rozloženie sa menilo na základe viditeľnosti jednotlivých častí, prípadne sa komponentám nastavil nový objekt triedy `RelativeLayout.LayoutParams` s upravenými pravidlami.

HTML stylizace textu v Androidu V Androide je možné vygenerovať formátovaný text z html jazyka pomocou triedy `Html` a funkcie `fromHtml`. Takto vytvorený text je

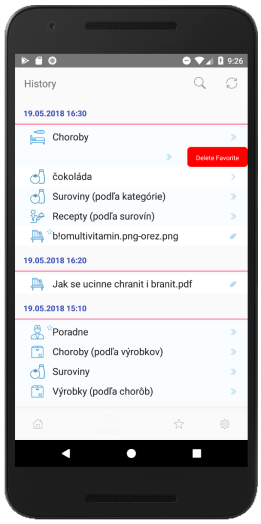
nastavovaný v bunke tabuľky prvku `TextView` pomocou funkcie `setText`. Príklad html formátovania môžeme sledovať na obrázku 6.17.

6.3.5 Označení potáhnutím řádku

Pre obe platformy je implementovaný mechanizmus na označenie obľúbenosti záznamu gestom potiahnutia riadku doľava. iOS používatelia sú na akcie gestom doľava zvyknutí z akcií zamknutej obrazovky ako aj vlastných natívnych aplikácii ako Notes. Pri Android platforme je gesto potiahnutia riadku do určitej strany v natívnych aplikáciách, ako napríklad Gmail vyhradené pre trvalejšie akcie ako vymazanie či archiváciu, ku ktorým je často krát implementované aj tlačidlo vrátenia akcie. Používatelia sú teda zvyknutí na tento pohyb ako akciu riadku a je im prirodzený. Na označenie obľúbenosti je použité potiahnutie riadku so zelenou farbou na pridanie a červenou na odobranie záznamu z obľúbených a následné potvrdenie stlačením odokrytého tlačidla. Po stlačení je zaslaná požiadavka na server o označenie pomocou webovej služby *set_mark* aj upravenie dát tabuľky.



Obr. 6.18: Označení oblíbenosti řádku v tabulce iOS (snímek obrazovky, vlastní)



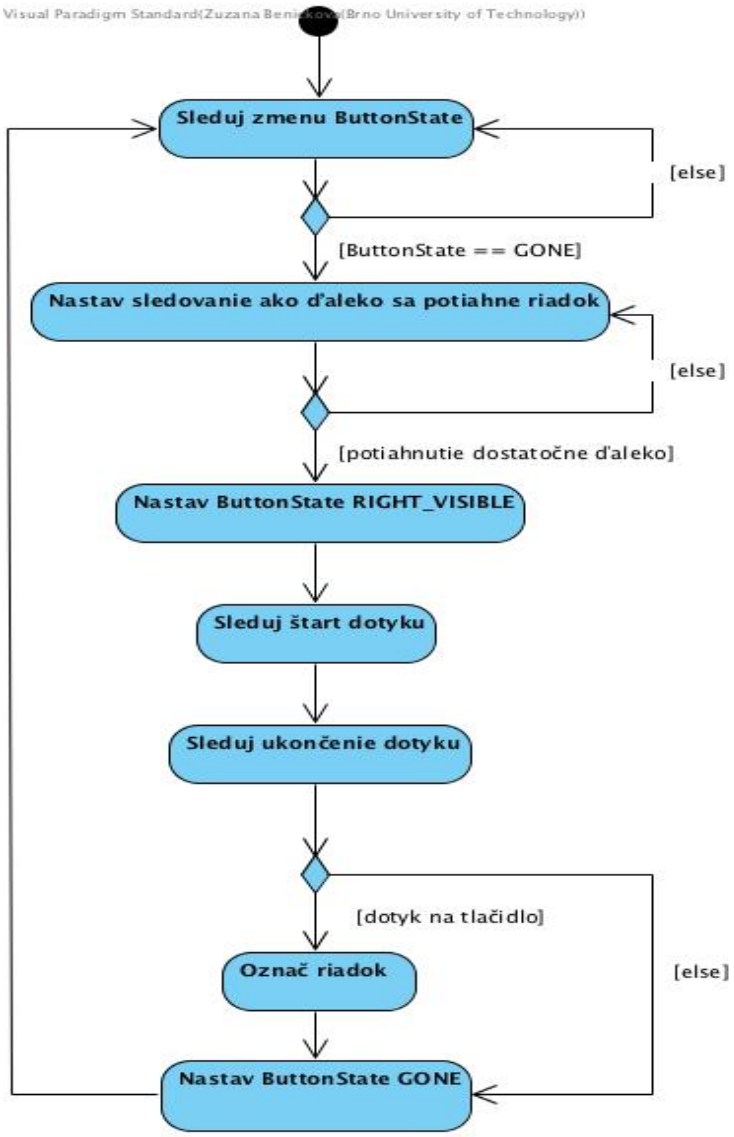
Obr. 6.19: Odznačení oblíbenosti řádku v tabulce Android (snímek obrazovky, vlastní)

Označení potáhnutím řádku v iOS

Delegát tabuľky ponúka implementovanie funkcie `editActionsForRowAt`, vďaka ktorej je možné pre jednotlivé riadky nastaviť ich spôsob upravovania. V triede `UITableViewRowAction` sa nastavuje text, farba aj následná akcia po stlačení potiahnutého tlačidla. Výsledný vzhľad môžeme sledovať na obrázku 6.18.

Označení potáhnutím řádku v Androidu

Pre zachytenie gesta potiahnutím riadku je použitá natívna trieda `ItemTouchHelper` a `ItemTouchHelper.Callback`. Trieda Callback danému riadku dovoľuje nastaviť pomocou funkcie `getMovementFlags` smer gesta, v ktorom sa bunka bude pohybovať a ItemTouchHelper



Obr. 6.20: Diagram aktivity tlačítka na potažení třídy SwipeController (zdroj vlastní)

pomocou funkcie `attachToRecyclerView` sleduje gestá na danej tabuľke. Následne sa pomocou triedy `OnTouchListener` odchytávajú konkrétne gestá a ich typ. Implementovanú logiku môžeme sledovať na diagrame aktivít triedy `SwipeController` 6.20. Môžeme pozorovať, že prístup je omnoho zložitejší ako v prípade implementácie iOS verzie. Výsledné zobrazenie môžeme vidieť na obrázku 6.19.

6.3.6 Nastavení pořadí řádků

Pre obe platformy je implementovaná možnosť zmeniť si poradie obľúbených položiek. V iOS verzií musí používateľ stlačiť horné tlačidlo na editáciu (dostupné iba v tabuľke 3 - Mark), aby sa dostal do editačného módu, v ktorom môže presúvať poradie riadkov, prípadne ich mazať. V Androide je možnosť presunutia riadkov po dlhom podržaní pozície v tabuľke 3 - Mark. Následne sa ukladá nové poradie na server pomocou služby `set_mark_all`. V iOS aplikácii sa funkcia volajúca túto službu volá po ukončení editačného módu, pri opätovnom stlačení tlačidla na editáciu. V Androide žiadny editačný mód nie je preto sa služba volá po uplynutí dvoch sekúnd bez zmeny poradia riadkov, kedy už môžem predpokladať, že používateľ skončil s editáciou.

Nastavení pořadí řádků v iOS

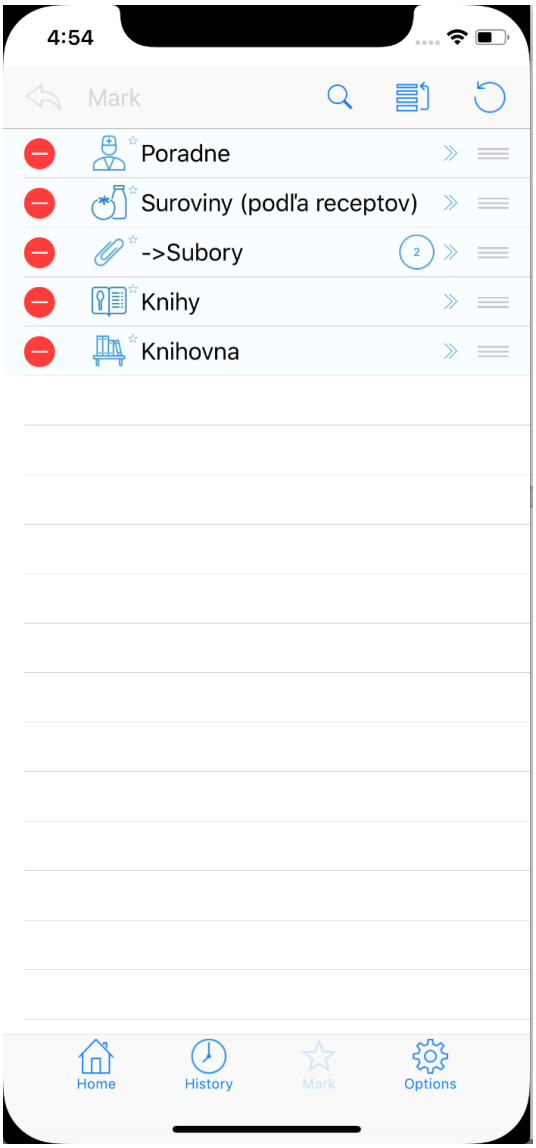
Pre umožnenie presunu riadkov v tabuľke sa opäť využívajú natívne možnosti delegáta tabuľky a implementuje sa funkcia `canMoveRowAt` a `moveRowAt`. Vďaka týmto funkciám nastavujem, že chcem v editačnom móde tabuľky, ktorú nastavujem pomocou funkcie `setEditing`, povoliť aj presúvanie riadkov a následne implementujem zámenu z predchádzajúcej na cieľovú pozíciu prvkov v poli dát. Tabuľka má v editačnom móde automaticky znázornené štandardné ikony na presúvanie prvkov a rovnako ikonu na zmazanie prvku ako môžeme sledovať na obrázku 6.21. Android žiadne automatické grafické znázornenie možnosti presunu nemá.

Nastavení pořadí řádků v Androidu

Pre nastavenie poradia v riadku sa rovnako ako v prípade označenia potiahnutím 6.3.5 využila trieda `Callback`. Keďže som danú funkcionalitu chcela implementovať iba v tabuľke obľúbených prvkov vytvorila som vlastnú triedu `DragController`, ktorú nastavujem iba pre tabuľku s obľúbenými hodnotami. Táto trieda reaguje na zachytené gestá pri dlhom potlačení riadku a volá funkciu pri zmene poradia hodnôt vo vlastnom adaptéry triedy `DragableAdapter`. DragableAdaptér následne implementuje reakciu na zachytené gestá zámenou prvkov v poli dát.

6.3.7 Vrchní lišta možností

Vrchná lišta slúži na zobrazenie aktuálnej pozície a možností v danom zozname. Logicky je lišta rozdelená na pravú a ľavú stranu. Ľavá strana slúži na navigáciu v rámci zanorenia v zoznamoch, ponúka možnosť tlačidla späť, aktuálny názov zoznamu ako aj možnosť stlačenia názvu na zobrazenie navigačného menu celého zanorenia 6.3.9. Pravá strana slúži na akcie s dátami aktuálneho zoznamu alebo súboru ako obnova dát, zapnutie vyhľadávania, zobrazenie detailu, zobrazenie aktuálneho sťahovania a zobrazenie možností externých aplikácii s daným súborom.



Obr. 6.21: Editační mód Mark tabulky iOS verze aplikace (snímek obrazovky, vlastní)

Vrchní lišta možností v iOS

V iOS vrchnú lištu implementuje trieda `UINavigationBar`. Do tejto triedy sú vkladané aktuálne požadované prvky UIBarButtonItem na základe typu aktuálneho zobrazenia. Tieto prvky je nutné rozdeliť na prvky ľavej a prvky pravej strany lišty. Pomocou funkcie `setRightBarButtonItems` a funkcie `setLeftBarButtonItems` sú dané prvky nastavované. Nevýhodou oproti práci s lištou v Androide je, že pole prvkov je definované ručne a treba riešiť otázku priority v poradí prvkov. V projekte boli vytvorené vlastné funkcie na manipuláciu s prvkami lišty a to `showFileButtons` a `hideFileButtons`, ktorá určuje, ktoré prvky pridať a odobrať pri zobrazení súboru. Využíva sa vlastná funkcia `addButtonRight`, ktorá určuje prioritu v poradí prvkov.

Vrchní lišta možností v Androidu

Lišta možností danej stránky je implementovaná komponentom Androidu `Toolbar`. Rovnako ako pre spodnú navigačnú lištu 6.3.8 sa jednotlivé komponenty `item` nastavujú v zdrojových súboroch `res/menu`. Následne sa v aktivite vo funkcii `onCreateOptionsMenu` nastavuje pomocou triedy `MenuInflater` zdrojový súbor. Aby si fragment podľa aktuálneho zoznamu nastavil dynamicky lištu, implementuje viditeľnosť jednotlivých prvkov vo funkcii `onPrepareOptionsMenu(Menu menu)`. Táto funkcia je zavolaná pokiaľ je aktuálne menu možností zrušené funkciou `invalidateOptionsMenu` a preto sa vždy vygeneruje nové, aktuálne požadované menu. Podobná funkcionalita v iOS chýba.

Indikace stahování na pozadí

Pretože samotné spracovanie dát je časovo mnoho krát náročné, je vyžadované od správne navrhnutej aplikácie, aby používateľ videl, že sa dáta načítavajú a aplikácia niečo na pozadí vykonáva. Indikátor je vložený do vrchnej lišty miesto tlačidla na obnovenie dát.

Indikace stahování na pozadí v iOS Pre zobrazenie priebehu sťahovania využívam v iOS triedu `UIActivityIndicatorView`. Následne pre spustenie animácie je volaná funkcia `startAnimating` prípadne `stopAnimating`. Na rozdiel od Androidu sa pracuje v iOS s dvoma stavmi objektu, hýbajúci sa a statický. Pomocou triedy `UIBarButtonItem` a možnosti inicializácie s vlastným `View` objektom je indikátor sťahovania vložený do hornej lišty a následne sa s ním pracuje ako s ostatnými prvkami lišty. Do lišty sa pridáva rovnako ako iné prvky pomocou funkcie `addButtonRight`, v ktorej je implementované, aby sa pozícia indikátoru nahradila s pozíciou tlačidla na obnovenie dát.

Indikace stahování na pozadí v Androidu V Anroide je na zobrazenie načítavania využitá trieda `ProgressBar`. Vytvorený objekt triedy ProgressBar je nastavovaný prvku `MenuItem`, ktorý je vložený v hornej lište a pomocou funkcie `setActionView` ho zobrazuje a následne vypína. Trieda ProgressBar poskytuje animované otáčanie bez nutnosti ho ďalej akokoľvek spustiť či vypnúť. ProgressBar teda nie je možné zobraziť bez animácie.

6.3.8 Spodní navigační lišta

V aplikácii spodná lišta slúži na navigáciu medzi tabuľkami *Home* - základné koreňové zobrazenie všetkých dostupných dát, prvotné volanie služby *get_zoznam* s *triedou* 0, *History*

- história prvotné volanie služby *get_zoznam* s *triedou* 1, *Mark* - označené položky prvotné volanie služby *get_zoznam* s *triedou* 2 a *Settings* - nastavenia.

Spodní navigační lišta v iOS

Spodnú navigačnú lištu v iOS prostredí implementuje trieda s názvom `UIToolbar`. Jednotlivé prvky tvoria rovnako ako prvky hornej lišty `UIBarButtonItem`. Pre správne rozpoloženie tlačidiel sa využíva typ tejto triedy nastavený na flexibilnú voľnú plochu, ktorá sa dynamicky prispôsobuje. Aby sa mohlo použiť tlačidlo s obrázkom aj požadovaným textom, bola rozšírená trieda `UIButton`. Trieda sa rozšírila o funkciu `alighVertical`. Tlačidlo s textom bolo možné zobraziť vďaka prvkom *titleEdgeInsets, imageEdgeInsets, contentEdgeInsets*, ktorým bolo možné priradiť vhodne nastavenú triedu `UIEdgeInsets`. Táto trieda určuje okraje zobrazovaného obdĺžnika daného view obejektu s posunom potrebným pre zobrazenie obrázka nad textom.

Spodní navigační lišta v Androidu

Práca so spodnou(rovnako ako s vrchnou) lištou je v Androide veľmi jednoduchá. Implementuje ju trieda `BottomNavigationView` a na jej naplnenie tlačidlami je v zdrojových súboroch potrebné definovať xml súbor v priečinku *res/menu*, v ktorom sú definované jednotlivé *item* prvky v rámci komponenty *menu*. Aktivita pomocou funkcie `findViewById(int id)` a odkazu na komponent `BottomNavigationView` nastavuje vzhľad navigačného menu. Taktiež aktivita implementuje reakciu na prechod do inej tabuľky pomocou triedy `OnNavigationItemSelectedListener`. Výhodou oproti iOS je, že danú ikonu s textom nie je nutné implementovať, ale funkcionalitu navigačná lišta ponúka sama.

6.3.9 Navigační strom

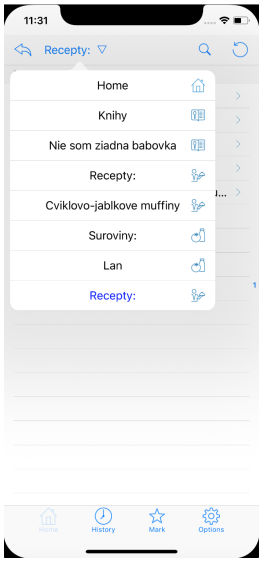
Navigačný strom slúži používateľovi na rýchlu navigáciu v zanorení. V oboch prípadoch som ho implementovala ako tabuľku s názvami a ikonami navštívených adresárov, ktorá sa zobrazí nad súčasnou polohou po stlačení názvu aktuálneho zoznamu. V oboch prípadoch je implementovaný na stlačenie názvu aktuálnej tabuľky, v prípade iOS je implementovaný aj na opakované stlačenie aktuálnej polohy v spodnej lište.

Navigační strom v iOS

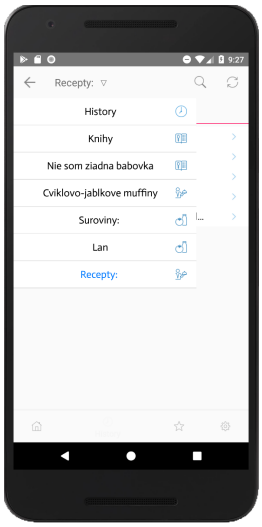
Pre zobrazenie navigačného stromu je vytvorená vlastná štruktúra v súbore *Main.storyboard*, v ktorom je definovaný vzhľad navigačného panelu. Trieda `PopoverViewController` zdedená z triedy `UIViewController` deleguje protokol pre spracovanie dát a akcií tabuľky. Danú inštanciu triedy napĺňame potrebnými údajmi, nastavujeme na typ zobrazenia ako modálny - menšie samostatne zobrazené okno nad aktuálnou obrazovkou a iniciujeme jeho zobrazenie funkciou `present`. Výsledný navigačný strom v iOS implementácii môžeme sledovať na obrázku 6.22.

Navigační strom v Androide

Pre zobrazenie navigačného stromu je rovnako ako v iOS vytvorená vlastná štruktúra - layout súbor *popup.xml*, v ktorom je definovaný vzhľad navigačného panelu. Navigačný strom je vytvorený pomocou triedy `PopupWindow`, do ktorej je vložená trieda `View` predstavujúca vytvorenú štruktúru. Po zavolaní funkcie `showAsDropDown(View)` sa objekt zobrazí



Obr. 6.22: Navigační strom iOS (snímek obrazovky, vlastní)



Obr. 6.23: Navigační strom Android (snímek obrazovky, vlastní)

nad aktuálnou aktivitou. Naplnenie tabuľky určuje trieda `PopUpAdapter`. Hlavná aktivita `MainActivity` podľa stlačeného tlačidla v aktuálnom `Fragment_page` vráti používateľa na požadované miesto. Výsledný navigačný strom v Android implementácii môžeme sledovať na obrázku 6.23.

6.3.10 Vyhledávání v datech

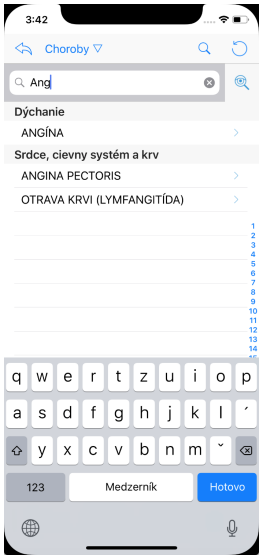
V aplikácii sú navrhnuté a implementované dva druhy vyhľadávania. *Lokálne vyhľadávanie* slúži na zobrazenie prvkov tabuľky, kde sa vložený text vyhľadáva v hodnotách *text* a *text2* v objektoch `APIReturn` tabuľky. *Serverové vyhľadávanie* prebieha pomocou webovej služby *get_search* na dodatočné potvrdenie vyhľadávania, v prípade iOS stlačenie ikony špeciálnej lupy, v Androide na potvrdenie po dopísaní textu.

Vyhledávání v datech v iOS

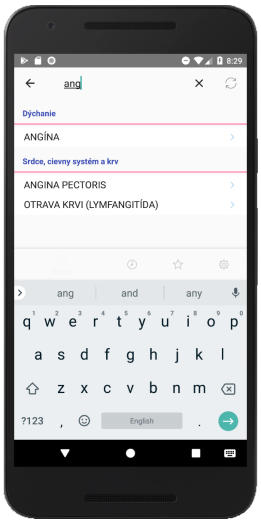
Delegovanie protokolu `UISearchBarDelegate` dovoľuje odchytiť akciu pri zmene textu vo vyhľadávacom poli funkciou `textDidChange`, následne upravuje zoznam filtrovaných prvkov pomocou funkcie `filter`. Vyhľadávacie pole je navrhnuté ako samostatná časť pod lištou možností, ktorú je tlačidlom možné zobraziť a znovu skryť. Pri zmene textu sa dynamicky mení aj zoznam aktuálnych dát.

Vyhledávání v datech v Androidu

V Androide je na vyhľadávanie použitá trieda `SearchView` a následne `OnQueryTextListener`. Tieto triedy, podobne ako v iOS implementácii, dovoľujú zachytiť akciu pri vložení textu. Rozdielna je poloha priestoru na vkladanie hľadaného textu - v iOS je pod hornou lištou, v prípade Androidu ju nahrádza. Jeho zobrazenie som na rozdiel od iOS nemusela implementovať, nastavila som prvku *item* v definovanom súbore *menu* hodnotu *actionViewClass* na triedu `SearchView` a vyhľadávacie pole sa zobrazilo po stlačení tlačidla automaticky.



Obr. 6.24: Zobrazené vyhledávaní iOS (snímek obrazovky, vlastní)



Obr. 6.25: Zobrazené vyhledávaní Android (snímek obrazovky, vlastní)

Následne je na odchytenú akciu zmeny textu filtrovaný zoznam sekcií adaptéru. Sekcie implementujú vďaka vytvorenému rozhraniu `FilterableSection` funkciu `filter`. Funkcia prechádza aktuálne dáta sekcie a vracia iba tie, ktoré v hodnote text a text2 obsahujú hľadaný reťazec.

6.3.11 Jazyková verze

Pre obe platformy je implementovaná jazyková verzia aplikácie. V prípade iOS verzie je možné jazyk nastaviť serverom. V prípade Android verzie je jazyk nastavený na základe jazyku systému mobilného zariadenia, nakoľko sa jeho vlastné nastavenie pomocou triedy `Location` neodporúča pre nekonzistentné chovanie. Ako nastaviť jazyk podľa nastavenia jazyka mobilu je rozobrané v oboch platformách. Implementovaná je anglická, česká, slovenská a maďarská verzia.

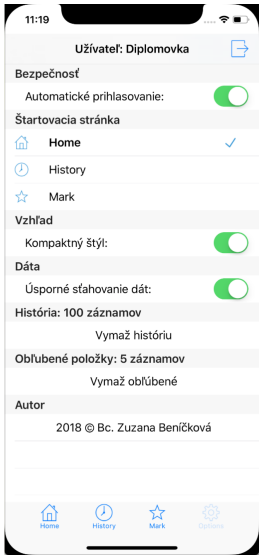
Jazyková verze v iOS

Xcode ponúka možnosť samostatného vytvorenia textovej verzie aplikácie. Sám vytvorí súbor pre zvolený jazyk, do ktorého sa vkladajú pre konkrétne texty preklady. Následne je na základe nastavenia v mobile nastavený zdrojový súbor textov. V implementovanej iOS verzii je jazyk pomocou poľa textov v rôznych jazykoch nastavený pomocou hodnoty *jazyk* v objekte `APIReturnList`. Na základné texty je implementovaný aj český, slovenský a maďarský jazyk.

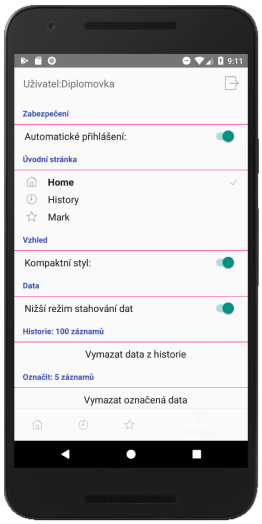
Jazyková verze v Androidu

V Android verzií je programátor prirodzene vedení k ukladaniu textov do zdrojového súboru `res/values/strings.xml`. Tento súbor je prednastavený na všetky nedefinované jazyky, tradične sa píše v angličtine. Pre vytvorenie jazykovej verzie sa tento súbor duplikuje s vlastnými prekladmi a názvom vo formáte: "string-*kód krajiny*".

6.3.12 Nastavení uživatele



Obr. 6.26: Nastavení aplikace iOS (snímek obrazovky, vlastní)



Obr. 6.27: Nastavení aplikace Android (snímek obrazovky, vlastní)

V aplikácii je štvrtá, pravá časť dolného navigačného menu aplikácie venovaná nastaveniu používateľa. Zahŕňa možnosti ako úvodná obrazovka pri spustení aplikácie, automatické prihlásenie, počet položiek histórie a obľúbených položiek, ich vymazanie a úsporné sťahovanie dát. Jej realizáciu na iOS zariadení môžeme sledovať na obrázku 6.26 a na Androide na 6.27.

Všetky tieto možnosti, okrem manipulácie s dátami histórie a obľúbených položiek sú nastavované pomocou perzistentných dát. Rovnako sa tu nachádza možnosť kompaktného zobrazenia dát, kedy je riadok tabuľky menší a zmestí sa viac dát na jednu obrazovku. Túto možnosť implementujem rovnako cez perzistentné dáta a pri nastavovaní vzhľadu a obsahu konkrétneho riadku beriem túto hodnotu ako rozhodujúcu pre určenie odsadenia a výšky riadku. Medzi používateľské nastavenia patrí rovnako ukladanie mena a hesla, na čo sa aplikácia opýta pred odhlásením. Automatické prihlasovanie je možné nastaviť v úvodnej obrazovke aj v nastaveniach aplikácie. Všetky dôležité alebo trvalé akcie používateľa, ako zapamätanie si mena a hesla, vymazanie histórie alebo obľúbených položiek musí používateľ dodatočne potvrdiť.

Obr. 6.28: Alert iOS (snímek obrazovky, vlastní)

Obr. 6.29: Toast Android (snímek obrazovky, vlastní)

Obr. 6.30: Alert Android (snímek obrazovky, vlastní)

Potvrzení a upozornení uživatele v iOS

V iOS je na potvrdenie rovnako ako upozornenie využitá trieda `UIAlertController`. V tejto triede je nastavovaný požadovaný nadpis, text aj štýl zobrazeného okna. Pre jednotlivé tlačidlá sa pomocou funkcie `addAction` pridávajú objekty `UIAlertAction`, ktoré definujú text tlačidla, vzhľad aj následnú akciu po stlačení. Aktuálny viewController tento objekt prezentuje funkciou `present`. Upozornenie pomocou okna, ktoré musí používateľ potvrdiť má výhodu v tom, že je zaručené, že si používateľ upozornenie všimne. Toto upozornenie v aplikácii využívam pri potvrdení udalosti ako odhlásenie z aplikácie, vymazanie zoznamov, upozornenie pri neúspešnom prihlásení, upozornenie pri vyžiadaní detailu súboru, ktorý neexistuje. Jedno z upozornení aplikácie môžeme vidieť na obrázku 6.28.

Potvrzení a upozornení uživatele v Androidu

V Androide je možné na upozornenie použiť dva spôsoby. Jeden je pomocou triedy `Toast`, ktorá používateľovi zobrazí na krátku dobu v spodnej časti obrazovky zvolený text. Trieda sa veľmi jednoducho používa, v projekte je vytvorená privátna inštancia tejto triedy a pomocná funkcia `showToast`, ktorá skontroluje aktuálne použitie objektu, pokiaľ je nenulový preruší ho pomocou funkcie `cancel`. Následne sa pomocou funkcie `makeText` a `show` daný toast zobrazuje. Použitie triedy toast je vhodné v prípade upozornenia alebo jednoduchej správy napríklad o úspešnosti určitej akcie. Použitie nie je vhodné, pokiaľ používateľ informáciu potrebuje pre ďalšie používanie aplikácie. Príklad použitia v aplikácii môžeme vidieť na obrázku 6.29. Využívam ho na informáciu pre používateľa o nedostupnom internete.
Pokiaľ je potrebné, aby používateľ s upozornením alebo výberom interagoval, alebo si ho určite všimol je použitá, podobne ako v iOS, trieda `AlertDialog.Builder`. V tejto triede, rovnako ako v triede `UIAlertController` je nastavovaný požadovaný nadpis, text správy a jednotlivé akcie pomocou triedy `DialogInterface.OnClickListener`. Pozíciu a štýl jednotlivých tlačidiel určuje spôsob, akým sú nastavené v rámci triedy `AlertDialog.Builder` a jeho funkcii `.setNeutralButton, .setPositiveButton, .setNegativeButton`. Výsledné použitie môžeme vidieť na obrázku 6.30. Alert využívam na podobné prípady ako v iOS a to odhlásenie, potvrdenie vymazania zoznamu či nesprávne meno a heslo.

6.3.13 Porovnání uživatelského rozhraní aplikace v iOS a Androidu

V rámci iOS je programátor viac vedený k využitiu dizajnu, ktorý už je implementovaný, kde naopak v prípade Androidu je programátor veľa krát (napríklad jazdec pri vyhľadávaní, prepínateľné tlačidlo) nútený určiť jednotlivé prvky a ich dizajn sám. V prípade tabuľky je v iOS verzii viac možností ako s tabuľkou a dátami pohodlne pracovať, naopak v Androide je práca s navigačnou a akčnou lištou jednoduchšia. Môžeme sledovať, že názvoslovie pre vrchnú a spodnú lištu je v iOS a Androide opačná. Predpokladám, že názvoslovie je na základe tradičného tlačidla späť v ľavom hornom rohu na iOS zariadeniach. V Androide nebolo nutné v spodnej lište implementovať odsadenie jednotlivých tlačidiel ani tlačidlo s textom, rovnako nebolo treba implementovať zobrazenie vyhľadávacieho poľa. Naopak v iOS som tieto možnosti implementovať musela. V Android implementácii na druhej strane chýba možnosť hlavičiek kategórii udržiavajúcich sa na vrchole tabuľky, rovnako informačný text pri potiahnutí tabuľky na obnovu dát.

Kapitola 7

Testování

V nasledujúcej kapitole sa zaoberám testovaním vytvorenej mobilnej klientskej aplikácie. V úvode kapitoly rozoberám navrhnuté testovanie, podmienky testovania a používateľský dotazník. V závere kapitoly následne rozoberám pozorované výsledky.

7.1 Návrh testovania

Pretože základom vytvorenej mobilnej aplikácie je dostupnosť na oboch platformách mobilných zariadení a vhodné používateľské prostredie a používateľský zážitok, rozhodla som sa výslednú aplikáciu testovať na skupine respondentov s ich osobnými zariadeniami. Na serveri je vytvorený testovací účet s menom *Diplomovka* a heslom *q*. Kvalitatívne testovanie prebehlo na 4 množinách respondentov: používatelia iOS a používatelia Android platformy ako primárneho zariadenia a používatelia oboznámení s aplikáciou a neoboznámení. Od respondentov sa očakáva základná zdatnosť používania mobilu. Testovala sa úspešnosť a čas zvládnutých pripravených úloh na vlastnej platforme, zobrazenie aplikácie na cudzej platforme a ich následné pozitívne a negatívne recenzie na obe verzie aplikácie. Základnými úlohami boli:

- Prihláste sa sa do aplikácie menom *Diplomovka* a heslom *q* tak, aby ste už nabudúce tento krok nemuseli robiť.
- Vyhľadajte všetky záznamy, v ktorých je Vaše krstné meno v zozname Knihovna.
- Pošlite mailom ľubovoľný obrázok(jpg, pdf) z História.
- Nastavte si úvodnú obrazovku po prihlásení na Históriu.
- V zozname Home označ položku Knihovna.
- Zmeňte poradie položky Knihovna v obľúbených Mark na posledné/prvé miesto.

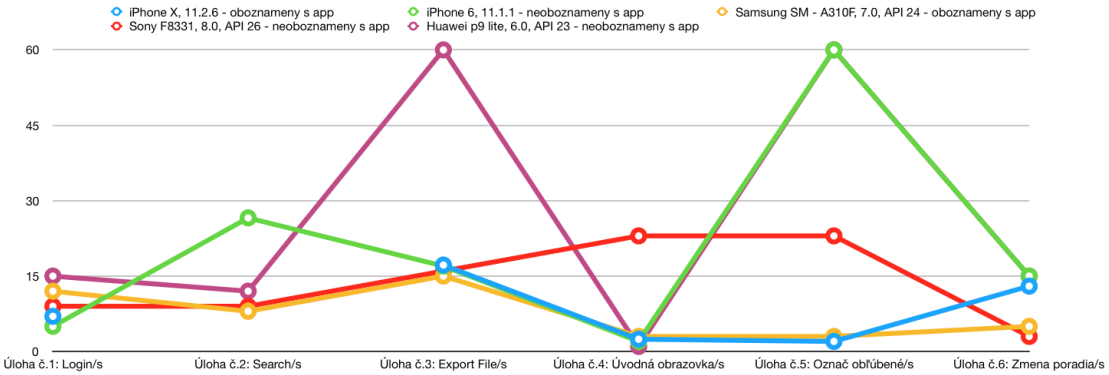
Okrem používateľského testovania dotazníkom a testovaní na ich vlastných zariadeniach, ktorých zoznam môžeme vidieť v prílohe C, podrobnejšie funkčné testovanie prebehlo na štyroch zariadeniach, na oboch platformách jedno mobilné zariadenie, jeden tablet. V nasledujúcej tabuľke môžeme sledovať konkrétne typy zariadení a číslo ich aktuálneho softvéru.

platforma - zariadenie	mobil	tablet
iOS	iPhone X, 11.2.6	iPad Pro, 10.3.1
Android	Nexus 5X, API 26	Samsung Galaxy Note 2014, API 23

Na každom zo zariadení som testovala jednotlivé funkcionality v priebehu implementácie, aj na záver v kompletnej aplikácii:

- prihlásenie používateľa
  - úspešné prihlásenie - načítanie dát
  - neúspešné prihlásenie - upozornenie o zadaní zlého mena alebo hesla a vrátenie do login aktivity
- prechod dátami všetkých typov
  - úspešný prechod - zobrazenie nových dát
  - neúspešný prechod - upozornenie o žiadnych načítaných dátach
- nastavenie vzhľadu riadku tabuľky
  - určenie maximálneho počtu riadkov - zobrazenie textu skráteného na požadovaný počet riadkov
  - html zobrazenie - zobrazené html formátovanie
  - nastavenie ikony
    - známa ikona - nastavenie ikony
    - neznáma ikona - nastavenie prednastavenej ikony
- automatické obnovovanie dát - v intervale obnovené dáta
- zobrazenie súboru
  - priblíženie, pohyb po súbore - úspešné priblíženie, pohyb po súbore na určené gesto
  - export do iných aplikácii mobilu - email, dátové úložisko
- vyhľadávanie v dátach
  - lokálne vyhľadávanie - nájdené prvky v danej tabuľke
  - serverové vyhľadávanie - načítané všetky prvky zo servera
- označenie/odznačenie riadku do obľúbených - riadok je označený/odznačený
- zmena poradia obľúbených položiek - riadky majú nové poradie
- nastavenie preferencii používateľa - chovanie aplikácie podľa nastavení

Testovanie hraničných odpovedí webových služieb ako prázdny zoznam, neznáma odpoveď či neúspešná odpoveď serveru som testovala počas implementácie webových služieb, reakciou je zobrazenie danej serverovej chyby pomocou alert 6.3.12 triedy. Jednotlivé výstupy by bolo možné testovať *unit testovaním* pomocou natívnej iOS knižnice *XCTest*. Testovanie serverovej strany aplikácie je ponechané na konkrétneho autora implementácie. Predpokladom je, že využívané služby sú otestované. Mobilná aplikácia je otestovaná na iOS zariadeniach so systémom 10.3.1 a vyššie a Android zariadeniach s API 22 a vyššie.



Obr. 7.1: Časové srovnání délky úkolů jednotlivých respondentů (zdroj vlastní)

7.2 Výsledky pozorovania

Z grafu jednotlivých časov úloh ako aj z poznámok respondentov som vyvodila nasledujúce pozorovania:

- Úloha č. 1 - respondenti nemali s touto úlohou problém, aplikácia na iOS reagovala priemerne rýchlejšie.
- Úloha č. 2 - serverové hľadanie v iOS aplikácii je spúšťané na stlačenie špeciálnej lupy, v Androide priestor na toto tlačidlo nebol, preto sa spúšťa na potvrdenie v klávesnici, toto riešenie sa nakoniec zdá omnoho lepšie nakoľko bolo toto tlačidlo pre respondentov prirodzenejšie - úloha im trvala menej - na základe tohoto pozorovania by som upravila v iOS verzii vyhľadanie tiež na potvrdenie z klávesnice.
- Úloha č. 3 - export súborov bol pre respondentov prirodzený, pri tejto úlohe sa vyskytol problém, že respondent s Android aplikáciou nevedel nájsť zoznam História, ktorý bol súčasťou úlohy, pretože v navigačnom menu nie je vidieť ikona aj text pokiaľ v zozname aktuálne používateľ nie je. Tento problém by sa dal vyriešiť prehľadnejšou navigáciou alebo krátkym úvodným tutoriálom k aplikácii pri prvom spustení, čo môžeme vidieť na malom čase respondentov, ktorí boli s aplikáciou oboznámení.
- Úloha č. 4 - priemerne prijateľný čas, žiadny z respondentov nemal s úlohou väčší problém.
- Úloha č. 5 - z tejto úlohy vidieť, že bolo pravdepodobne zvolené nesprávne gesto na označenie obľúbenosti. Dvaja z piatich respondentov neboli schopní dokončiť úlohu bez nápovedy. Rovnako v grafe môžeme sledovať, že po oboznámení používateľa s aplikáciou nemali respondenti s úlohou problém. Rovnako by sa tento problém vyriešil krátkym úvodným tutoriálom k aplikácii pri prvom spustení
- Úloha č. 6 - Android má natívny proces presunu riadkov prirodzene rýchlejší.
- Preferencie - všetkým respondentom sa páčil vzhľad iOS verzie viac

Z uvedených pozorovaní by som do budúcna v aplikácii vylepšila:

- Android: spodná lišta výraznejšia, text tlačidla by bol zobrazovaný vo všetkých stavoch.

- iOS: potvrdenie vyhľadávania aj na stlačenie potvrdenia v klávesnici
- Pridanie úvodného navigačného tutoriálu po prvom spustení
- Prípadná zmena gesta na označenie obľúbenosti
- Prípadná zmena dizajnu Android aplikácie

Kapitola 8

Záver

V projekte som sa úspešne oboznámila s možnými spôsobmi vývoja aplikácii na mobilné zariadenia 2 a detailne sa zamerala na natívny vývoj aplikácii pre iOS 3 a Android 4 zariadenia. Priblížila som podobnosti aj rozdiely implementácií rôznych platforiem. V projekte je navrhnutý univerzálny spôsob zobrazovania dát zo servera pomocou webových služieb. Môžeme pozorovať, že iOS aj Android platforma ponúka riešenia vyžadovanej problematiky.

Projekt definuje základné prístupy ku vyvíjaniu na mobilné platformy, následne rozoberá ich najväčších gigantov. V prípade iOS môžeme prirodzene pozorovať, že používateľ je natívnymi možnosťami prostredia ako implementované indexovanie pre sekcie tabuľky, úpravy riadkov tabuľky či práca s webovým rozhraním vedený k aplikácii prirodzene spĺňajúcej štandard iOS vzhľadu. Android naopak prirodzene vedie programátora k udržiavaniu členitého kódu roztriedeného na triedy určených funkcií. Delegovanie protokolov v rámci iOS vývoja výrazne zrýchľuje implementáciu, pretože vieme hlavnú funkcionalitu sústreďovať na jedno miesto a definuje sa iba trieda pre vzhľad riadka tabuľky. V Androide je napríklad potrebné definovať vlastnú triedu adaptéra tabuľky a rovnako riadku. Môžeme sledovať množstvo podobností v oboch platformách ako rozvrhnutie aplikácie, Aktivita predstavuje CotrollerView, súbor info.plist v iOS je definičný súbor AndroidManifest.xml, trieda Fragment je ContainerViewController, trieda Intent je podobná triede Segue. Jednotlivé prístupy majú aj svoje rozdiely, Android je prirodzene prácnejší ako iOS, so súčasným jazykom Java je programátor nútený opakovať množstvo redundantného kódu, ako napríklad ošetrenie uloženia všetkých aktuálnych hodnôt pri otočení obrazovky. Rovnako je nutné pri implementácii brať do úvahy rôzne API s rôznymi funkčnými vlastnosťami jednotlivých zariadení. Zásadne slabšími komponentami, ktoré som pri Android vývoji cítila bol webový prehliadač a práca s tabuľkou, no naopak poskytoval omnoho väčšiu voľnosť v návrhu vzhľadu. V iOS implementácii bola práca s hornou aj spodnou lištou náročnejšia ako v Androide a rovnako výrazne chýbala možnosť distribúcie aplikácie pomocou inštalačného súboru, ako je to možné v Androide. Z testovania 7 môžeme pozorovať, že iOS aplikácia je napriek rovnakej implementácii prirodzene plynulejšia a svojím štandardným vzhľadom sa používateľom viac páči.

Navrhla som a implementovala klient-server mobilnú aplikáciu na dvoch najpoužívanejších platformách mobilných zariadení iOS a Android. Aplikácia slúži ako univerzálny prehliadač unifikovaných dát a dokumentov, ktorý pracuje s navrhnutými webovými službami s požadovaným vstupom a výstupom. Serverová časť je vďaka aplikácii absolútne nezávislá a schopná dáta používateľovi prehľadne sprístupniť a nastaviť vzhľad a funkcionalitu podľa vlastných požiadaviek. V oboch platformách som implementovala jednoduchú prácu s dátami, prácu s webovými službami, tabuľkové rozhranie, webové rozhranie,

prácu so súbormi, používateľské prispôsobenie a vyhľadávanie nad dátami 6. Navrhnuté riešenie ponúka prístupný prehľad ľubovolných dát v mobilnej aplikácii a je jednoducho implementovateľné. Vďaka univerzálnemu návrhu je možné zobraziť dáta z rôznych systémov používateľovi v jednej aplikácii bez nutnosti implementácie mobilnej aplikácie. Aplikáciu je možné prispôsobiť na jednoduchý prehľad ako aj komplexný dokumentový systém. Vďaka svojej univerzálnosti môže byť nápomocná pri projektoch a výskumoch v ktorých je sústredený návrh na zbieranie dát a serverovú časť. Mobilná aplikácia môže slúžiť ako doplnkový testovací nástroj, zobrazovať aktuálne stavy systémov, môže pomôcť pri prezentovaní výsledkov alebo ku sprístupneniu systému používateľom.

Pri opätovnom návrhu projektu by som zlepšila štruktúru iOS aplikácie a implementovala logiku Fragmentov - ContainerControllerView aj v rámci iOS riešenia, v prípade Androidu by som v budúcnosti zvolila jazyk Kotlin. Projekt by sa dal ďalej rozšíriť o možnosť jednoduchého prehľadu unifikovaných grafových dát ako som naznačila v prílohe D. Tiež by bolo zaujímavé rozšíriť funkcionalitu o viac aktívnych akcií používateľa, ako vkladanie súborov na server či interakcia s aktuálnymi dátami.

Literatúra

[1] Barea, A.; Ferre, X.; Villarroel, L.: *Android vs. iOS Interaction Design Study for a Student Multiplatform App*. 2011, [Online; navštíveno 14.01.2018].
URL
https://link.springer.com/content/pdf/10.1007%2F978-3-642-39476-8_2.pdf

[2] informačních technologií VUT v Brně, F.: *Diplomové práce*. [Online; navštíveno 22.05.2018].
URL http://www.fit.vutbr.cz/study/DP/DP.php

[3] Dave.B: *Android timer? How-to?* STACK OVERFLOW, Leden 2011, [Online; navštíveno 14.05.2018].
URL https://stackoverflow.com/questions/4597690/android-timer-how-to

[4] Developers, A.: *Layouts*. [Online; navštíveno 14.01.2018].
URL https://developer.android.com/guide/topics/ui/declaring-layout.html

[5] Goadrich, M. H.; Rogers, M. P.: *Smart smartphone development: iOS versus android*. 2011.

[6] Gogetap, B.: *StickyHeaders*. [Online; navštíveno 19.05.2018].
URL https://github.com/bgogetap/StickyHeaders/blob/master/README.md

[7] Heitkötter, H.; Hanschke, S.; Majchrzak, T. A.: *Evaluating Cross-Platform Development Approaches for Mobile Applications*. Springer, Berlin, Heidelberg, 2012, ISBN 978-3-642-36607-9.

[8] Inc., A.: *Human Interface Guidelines iOS*. 2018, [Online; navštíveno 14.01.2018].
URL https:
//developer.apple.com/ios/human-interface-guidelines/overview/themes/

[9] Keur, C.; Hillegass, A.: *iOS Programming: The Big Nerd Ranch Guide (6th Edition) (Big Nerd Ranch Guides)*. Big Nerd Ranch Guides, 2017, ISBN 978-0134682334.

[10] L4 Digital, a. G. c.: *FastScroll*. [Online; navštíveno 19.05.2018].
URL https://github.com/L4Digital/FastScroll

[11] Lattner, C.: *Chris Lattner's Homepage*. [Online; navštíveno 14.01.2018].
URL http://nondot.org/sabre/

[12] Pagani, G.: *SectionedRecyclerViewAdapter*. [Online; navštíveno 19.05.2018].
URL https://github.com/luizgrp/SectionedRecyclerViewAdapter

[13] Philips, B.; Stewart, C.; Marsicano, K.: *Android Programming: The Big Nerd Ranch Guide (3rd Edition) (Big Nerd Ranch Guides)*. Big Nerd Ranch Guides, 2017, ISBN 978-0134706054.

[14] Rivest, R.: *The MD5 Message-Digest Algorithm*. MIT Laboratory for Computer Science and RSA Data Security, Inc., Duben 1992, [Online; navštíveno 14.05.2018].
URL https://tools.ietf.org/html/rfc1321

[15] Vincent, J.: *99.6 percent of new smartphones run Android or iOS*. [Online; navštíveno 14.01.2018].
URL https://www.theverge.com/2017/2/16/14634656/android-ios-market-share-blackberry-2016

[16] Williams, O.: *Apple announces Swift, a new programming language for iOS and OS X*. [Online; navštíveno 14.01.2018].
URL https://thenextweb.com/apple/2014/06/02/apple-announces-swift-new-programming-language-ios/

Přílohy / Appendices

Príloha A

Obsah přiloženého CD

Priložený CD disk k práci obsahuje nasledujúce súbory:

- Screenshots/ – uchováva snímky oboch vytvorených mobilných aplikácii,
- TechnickaSprava/ – uchováva všetky potrebné dokumenty ku vytvoreniu tejto správy
- ZdrojoveSoubory/
  - Android/ – obsahuje príslušné zdrojové súbory k Android aplikácii, DP_xbenic03.apk inštalačný súbor Android aplikácie a README súbor k návodu na inštaláciu
  - iOS/ – obsahuje príslušné zdrojové súbory k iOS aplikácii a súbor README
- README – textový súbor s popisom obsahu priloženého CD,
- TechnickaSprava.pdf – súbor vo formáte .pdf obsahujúci túto správu

Príloha B

Manuál

V úvode je potrebné sa do aplikácie prihlásiť. Používateľ do prvého políčka vkladá meno, do druhého požadované heslo. (Diplomovka/q) Pomocou prepínacieho tlačidla je možné pri najbližšom spustení aplikácie tento krok preskočiť. Toto nastavenie ostane ale iba v prípade, že sa prihlásenie podarilo.

Následne má používateľ možnosť prechádzať medzi troma tabuľkami dát pomocou stlačenia tlačidla spodnej lišty. V prvej vidí všetky svoje dostupné dáta, v druhej históriu prechodov, v tretej označené položky zaslané serverom. Po stlačení riadku tabuľky sú načítané nové dáta, ktoré sú v dokumentovej hierarchii vnorené. Môžeme prejsť do tabuľky alebo na súbor. V prehľade tabuľky máme možnosť potiahnuť tabuľku smerom dole pre jej obnovenie dát. Jednotlivé riadky je možné potiahnuť do ľavej strany pre ich označenie a odznačenie obľúbenosti.

Aplikácia ponúka možnosť využiť tlačidlá vrchnej lišty s funkciami menovanými zľava a to: ísť späť v hierarchii(ak nie sme v koreni adresára) - tlačidlo šípky späť, otvoriť navigačný strom - tlačidlo názvu tabuľky, vyhľadať v dátach - tlačidlo lupy a obnoviť dáta - tlačidlo šípok v kruhu.

V prehľade súboru môžeme vykonávať nasledujúce funkcie menované zľava a to: ísť späť v hierarchii(ak nie sme v koreni adresára) - tlačidlo šípky späť, otvoriť navigačný strom - tlačidlo názvu tabuľky, export súboru a detail súboru.

V nastaveniach aplikácie je možné nastaviť nasledujúce funkcie: nastavenie automatického prihlasovania prepnutím prepínača, nastavenie úvodnej stránky stlačením tlačidla, nastavenie kompaktného zobrazenia prepnutím prepínača, nastavenie malého objemu sťahovaných dát prepnutím prepínača, vymazanie histórie, vymazanie obľúbených položiek stlačením daného riadka tabuľky.

V štvrtej časti je možné nastaviť si osobné preferencie a odhlásiť sa. Pri odhlásení je možnosť zapamätania si mena a hesla stlačením tlačidla OK v upozornení pri odhlasovaní.

Príloha C

Výsledný dotazník testování aplikace

Počas kvalitatívneho testovania sme s respondentmi prešli niekoľko základných úloh na otestovanie prvkov aplikácie a vyplnili dotazník ohľadom ich pocitov z aplikácie a preferencie aplikácie na oboch platformách. Polovica účastníkov bola oboznámená s aplikáciu, druhá nie. Úlohy mali nasledujúce znenie:

- Prihláste sa sa do aplikácie menom *Diplomovka* a heslom *q* tak, aby ste už nabudúce tento krok nemuseli robiť.
- Vyhľadajte všetky záznamy, v ktorých je Vaše krstné meno v zozname Knihovna.
- Pošlite mailom ľubovoľný obrázok(jpg, pdf) z História.
- Nastavte si úvodnú obrazovku po prihlásení na Históriu.
- V zozname Home označ položku Knihovna.
- Zmeňte poradie položky Knihovna v obľúbených Mark na posledné/prvé miesto.

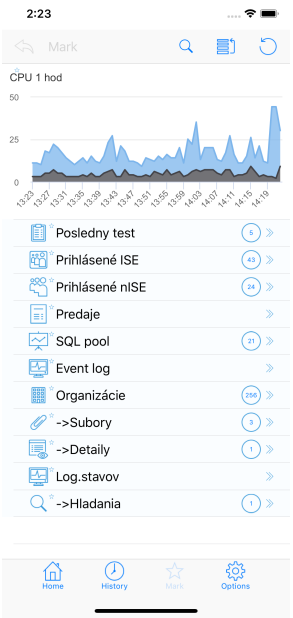
Výsledná tabuľka je nasledovná:

platforma - znalost aplikácie	Úloha č.1: Login/s	Úloha č.2: Search/s	Úloha č.3: Export File/s	Úloha č.4: Úvodná obrazovka/s	Úloha č.5: Označ obľúbené/s	Úloha č.6: Zmena poradia/s	Pocit z aplikácie vlastnej platformy	Ktorá aplikácia je Vám sympatickejšia?	Poznamky hodnotitela:
iPhone X, 11.2.6 - oboznameny s app	7	13,26	17,20	2,44	2	13	Super	iPhone	Search - az na stlacenie tlacidla - nevsimol si ho najskor
iPhone 6, 11.1.1 - neoboznameny s app	5	26,58	17	2,0	60	15	Spodok v androide zle vidno	iPhone, krajsi dizajn	Nevidel , search, stlacaju primarne hotovo a nie tlacidlo. Trebalo povedat ako oznacit.
Samsung SM - A310F, 7.0, API 24 - oboznameny s app	12	8	15	3	3	5	Pekna, menej plynula	iPhone	-
Sony F8331, 8.0, API 26 - neoboznameny s app	9	9	16	23	23	3	Prehladna -musi sa v tom otukat, nieje hned jasne co treba robit	iPhone ma lepsiu odovu	Kvoli casovej tiesni si v settings nevsimla nastavovanie a hladala ho. Oblubene hladala.
Huawei p9 lite, 6.0, API 23 - neoboznameny s app	15	12	60	1,0	60	15	Dobra	iPhone	Nevedel najst historiu v navigacii dole niesu napisy. Trebalo povedat ako oznacit riadok.

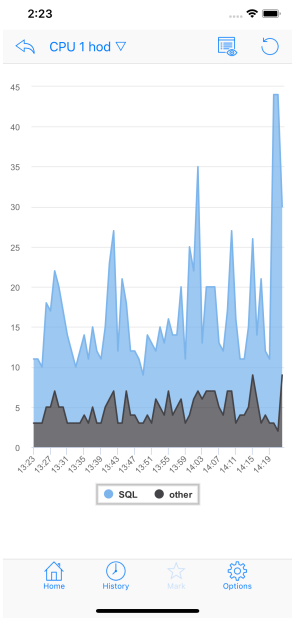
Obr. C.1: Výsledná tabuľka dát nazbieraných respondentov. (zdroj vlastní)

Príloha D

Návrh práce s grafovou knižnicou v iOS



Obr. D.1: Grafové zobrazení v tabulce (sní-



Obr. D.2: Grafové fullscreen zobrazení (sní-
mek obrazovky, vlastní)

mek obrazovky, vlastní)

V iOS prototype aplikácie bol implementovaný aj univerzálny prehľad grafov. Nakoľko táto časť v Androide implementovaná nie je, v práci túto oblasť a jej odlišnosti implementácie nerozoberáme. Jedná sa o využitie webovej služby *get_graf* , ktorá v unifikovanej podobe zasiela dáta na naplnenie konkrétneho určeného typu grafu. Pre položku typu *Graf* je navrhnutá vlastná štruktúra v triede `ViewController` nazvaná `DataGraf`. Táto trieda vychádza zo štruktúry zaslanej webovým serverom ako odpoveď služby get_graf. Pre implementáciu bola použitá externá knižnica `Highcharts`, z ktorej štruktúra dát tiež silno vychádza, aby bolo použitie čo najjednoduchšie. Graf môže byť zobrazený priamo v tabuľke (pokiaľ je hodnota `rozbaleny` nastavená v objekte triedy APIReturn) vo funkcii `cellForRowAt`, prípadne je zobrazený na celú obrazovku až na stlačenie konkrétneho políčka. Graf je vytváraný vo vlastnej funkcii `createGraf` pomocou triedy exterej knižnice `HIGChartView`.

Danou implementáciou sme pripravili univerzálny nástroj na zobrazenie akýchkoľvek dát zo strany servera pomocou grafového zobrazenia. Implementované graf v rámci tabuľky môžeme sledovať na obrázku D.1 a následne rozložený graf na celú obrazovku na obrázku D.2.